USING WRITE BUFFERS IN SYSTOLIC ARRAY ARCHITECTURES TO MITIGATE THE NUMBER OF MEMORY ACCESS PRODUCED BY ROW STATIONARY DATAFLOWS

An Undergraduate Research Scholars Thesis

by

DANIEL U. PERALTA VELAZQUEZ

Submitted to the LAUNCH: Undergraduate Research office at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by
Faculty Research Advisor/s: Paul V. Gratz
Kevin Nowka

May 2021

Major: Computer Engineering

RESEARCH COMPLIANCE CERTIFICATION

Research activities involving the use of human subjects, vertebrate animals, and/or biohazards must be reviewed and approved by the appropriate Texas A&M University regulatory research committee (i.e., IRB, IACUC, IBC) before the activity can commence. This requirement applies to activities conducted at Texas A&M and to activities conducted at non-Texas A&M facilities or institutions. In both cases, students are responsible for working with the relevant Texas A&M research compliance program to ensure and document that all Texas A&M compliance obligations are met before the study begins.

I, Daniel U. Peralta Velazquez, certify that all research compliance requirements related to this Undergraduate Research Scholars thesis have been addressed with my Research Faculty Advisor/s prior to the collection of any data used in this final thesis submission.

This project did not require approval from the Texas A&M University Research Compliance & Biosafety office.

TABLE OF CONTENTS

ABSTRACT

Using write buffers in Systolic Array architectures to mitigate the number of memory access produced by Row Stationary dataflows

Daniel U. Peralta Velazquez
Department of Electrical and Computer Engineering
Texas A&M University

Research Faculty Advisor: Paul V. Gratz
Department of Electrical and Computer Engineering
Texas A&M University

Research Faculty Advisor: Kevin Nowka
Department of Electrical and Computer Engineering
Texas A&M University

New applications of Deep Neural Networks are being designed such as fraud detection [1], short term weather precipitation forecasts [2], and cancer prognosis prediction [3]. Nonetheless, their respective models are getting more complex with an increasing number of depth layers. These models require millions of computations that conventional CPU and GPU architectures will take a significant amount of computational time. The data distribution of these models is well known; they mostly consist of dot product operations between inputs and filters. Applications such as self-driving cars required fast response time and accurate predictions. Current research [4][5] introduces accelerator architectures based on 2D systolic arrays as they provide high efficiency in performing multiplication and accumulation operations. Computational and power cost define performance, memory accesses attribute the highest cost to current architecture models [6]. In order to enhance the performance of DNN accelerators, parallelism is extracted by breaking convolution into partial computations at the expense of segmenting output memory accesses. This thesis ex-

plores the implementation of an accumulator microarchitecture component based on column pipelined adder trees with the purpose of collecting and aggregating output computed values based on destination address. The results of this work showed a 3.3x and 2.15x speedup for Tiny-YOLO and AlexNet CNN using a 32x64 Systolic Array. Through the reduction of computed values developers will be able to explore novel data mappings to extract parallelism based on data locality.

DEDICATION

Por ustedes abuelitos que me dieron la oportunidad de seguir mis sueños.

Madre y padre ah pasado demasiado en los ultimos años de nuestras vidas y estoy agradecido por su apoyo y amor incodicional.

Papa ahora entiendo el esfuerzo y parte de los obstaculos que enfrentaste en tus estudios. Es un gran sacrificio pero el aun mas fue dejar lo que construiste para buscar una mejor vida para nosotros, siempre estare agradecido.

Mama esos duros tiempos donde el desvelarse no era una opcion sino una obligacion. Me demostraste que la vida es dura pero el esfuerzo vale la pena y por ahora este esfuerzo es testimonio del valor de nuestros logros, gracias por tu apoyo incondicional.

Judith, la reyna de la casa y la persona que mas se simila con las diffcultades y logros de mi vida, gracias por seguir luchando en un pais que se mostro dificil a el inicio, sigue tus sueños mi architecta preferida confio en tu capacidad.

Carnalitos, Juan Diego y Pepe, tengo grandes esperanzas en ustedes. Espero y parte de mi trabajo y logros sirvan de aspiracion para los suyos. REcuerden que la satisfacion y el exito no se mide con dinero ni poder sino con la oportunidad de vivir en felicidad contigo y los que te rodean.

To my friends in school, Mexico and the rest of the US. Thank you guys for the unconditional support. Is quiet easy to thank you when everything is going good lmao but certainly is not easy to be there when things turn difficult. Thank you HijuePhD for the constant love, support, and hype they have helped me tremendously. Shout out to Algae and Jess for submitting their thesis and encouraging mine.

Roomies (Ed, Mando, Chris) thank you for tolerating a crazy lockdown time period with me and for allowing me to talk about my rising frustrations with this work.

Lastly, thank you to all of you that took a minute to understand my work and hear me rant without telling me to stop (CMSA, TCCA, LCAA, Latilo, LSAMP). You are a real one and likely reading this piece. I try to personalize my dedication messages but I have 5 minutes to submit, tarde como siempre.

Los quiero como no se imaginan y estoy feliz y agradecido de haber cruzado caminos en la vida.

ACKNOWLEDGMENTS

Contributors

I would like to thank my faculty advisor, Dr. Paul V. Gratz, Dr. Kevin Nowka and my graduate mentor, Cesar Lopez Carrasco, for their guidance and support throughout the course of this research.

Thanks also go to my friends and colleagues and the department faculty and staff for making my time at Texas A&M University a great experience.

Finally, thanks to their LSAMP Program for their encouragement and to Dr. Samuel Merriweather for his patience and advice.

The simulation tools used for "Using write buffers in Systolic Array architectures to mitigate the number of memory access produced by Row Stationary dataflows" were provided by Cesar Lopez Carrasco. The analyses depicted in Using write buffers in Systolic Array architectures to mitigate the number of memory access produced by Row Stationary dataflows" were conducted in part by Cesar Lopez Carrasco and were published in 2021 International Symposium on Computer Architecture.

All other work conducted for the thesis was completed by the student independently.

NOMENCLATURE

CNN	Convolutional Neural Network
DNN	Deep Neural Network
HPRC	High Performance Research Computing
FLOAT	Floating Point
PE	Processing Element
SAS	Systolic Array Simulator
IPC	Intructions per Cycle
ISA	Instrucion Set Architecture

1. INTRODUCTION

1.1 Convolutional Neural Networks

Convolutional Neural Networks are a type of Neural Networks often used for image recognition through raw data classification. Frank Rosenblatt, introduced the idea of the perceptron as a predicate that can be represented as whether certain subsets of points are a member of a family of classes [7]. In brief, the perceptron serves as the building block classifier that gave rise to many Artificial Intelligence applications. Convolutional Neural Networks are composed of several layers represented as numerical matrices. These layers are formed of filters and convolutional feature outputs with the first layer being the input data Figure 1.1. Filters are designed to identify a specific feature from the raw data and are typically smaller in dimensions. The main interaction between layers consists of hovering the filter in the data layer and compute a convolution operation for every data point Figure 1.2. A convolution involves computing the dot product between the filter and data matrices to produce an output feature layer. The number of convolutions needed increase rapidly as the number of filters and the size of input data increase.

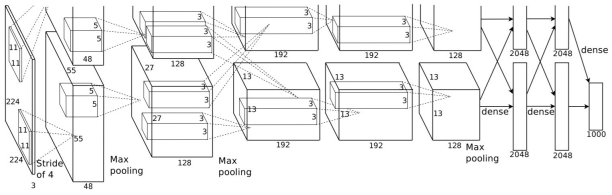

Figure 1.1: AlexNet CNN Model ©Alex Krizhevsky

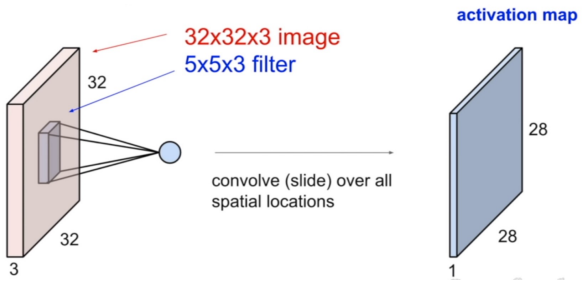

Figure 1.2: Convolutional Layer ©Fei-Fei Li

1.2 GPU Architecture

The large amount of computations and dataflow of a CNN requires dedicated hardware capable of parallelizing convolutional operations. Previous work by Alex Krizhevsky presents one of the first modern image recognition CNN. In order to achieve significant results a highly optimized GPU implementation of 2D convolution was written to properly train the respective CNN. Additionally, two GPUs were used due to memory limitations and the training lasted between five and six days to be completed [8]. GPUs are vector machines that can compute and parallelize data operations at higher computational speeds due to the independence and homogeneity of the data operations. GPUs are particularly useful in image processing because of the ability to apply a linear transformation to 3D images and can be re-purposed via General Programming for GPU to calculate a convolution of a matrix [4]. However while capable, adaptable GPUs are not designed to train CNNs and more dedicated hardware is needed to make operations more efficient and attain higher performance.

1.3 TPU Architectures and Systolic Arrays

Systolic Arrays consists of a set of interconnected Processing Elements, capable of performing a simple operation [9] Figure 1.3. Current research proposes Systolic Arrays based architectures because of the ability to perform multiply and accumulate (MAC) operations, and reuse and share data between PEs [4][5]. There are many dataflow implementations that position filters, outputs, and input elements differently in these architectures; Output Stationary, Input Stationary,

and Weight Stationary. These implementations have different design trade offs including memory accesses, data re utilization, and output reductions. Ongoing research is being conducted to find an optimized data mapping depending on the characteristics of the neural network model.

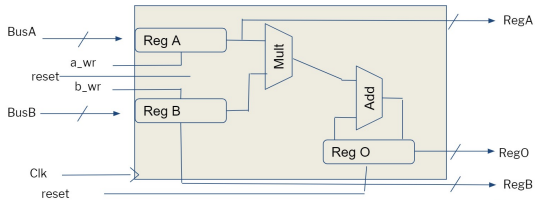

Figure 1.3: Processing Element ©Lopez Carrasco

1.4 Accelerators Data Movement

Convolutional Neural Networks perform heavy data movement. According to Tushar Krishna [6], memory accesses highly limit the performance of current hardware and ALU operation costing x1, PE x3, and DRAM access of x500. The properties of a Systolic Array allow temporary storage and access to other PE of partial data accumulations for a specific data point without constantly accessing the main memory hierarchy. However, after a convolution is computed the resulting value needs to be sent to memory in order to clear the Systolic Array and begin with the next convolution. In order to reduce direct access to DRAM, a memory buffer or scratchpad needs to be implemented and unlike conventional CPU instructions the data is well known and there is no need for a cache hierarchy. A buffer system will decrease memory accesses and increase the performance of future accelerators architectures.

2. METHODS

The main focus on reducing memory accesses consisted of reducing convolution computed values by aggregating values belonging to the same address on memory. Several design implementations were explored and are discussed in the following subsections to take advantage of data locality and re utilization.

2.1 Full Architecture: Comparison and changes

Previous accelerator architectural designs consist of exploiting parallelism from convolutions at the expense of segmenting memory due to the heavy flow of data from the main computing unit. This is applicable to row-wise dataflows where data locality allows to exploit parallelism at a higher degree. The previous CNN accelerator architecture as shown in Figure 2.1 consist of a set of feeding registers to populate the main computing unit. The main computing unit is composed of a Systolic Array of Processing Elements and post processing of data takes place at a collection and write out buffer where several computing cycles are used to write back to main memory.

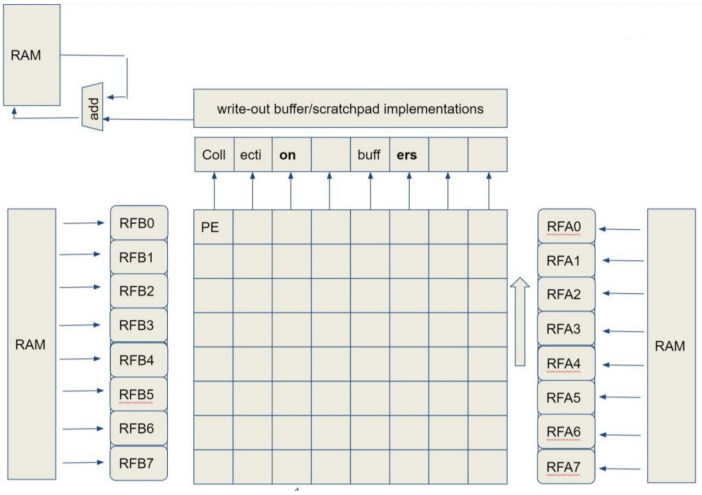

Figure 2.1: Previous Accelerator Architecture ©Cesar Lopez Carrasco

As part of reducing main memory accesses this work proposed system level and microarchitecture design changes in order to maintain a steady dataflow of processed data. The proposed changes take place in the post processing of data after the Systolic Array computes the corresponding FLOAT adding and dot product operations between layers.

In early stages of development adder trees column-wise were implemented to directly mapped every PE output to the inputs of the adder tree. The early stage of development as shown in Figure 2.2 produced promising results through a significant reduction of computing cycles. Despite the reduction of computing clock cycles two main issues arise from this design implementation; hardware expensive components (expensive to scale up in input size) and under utilization of adder trees.

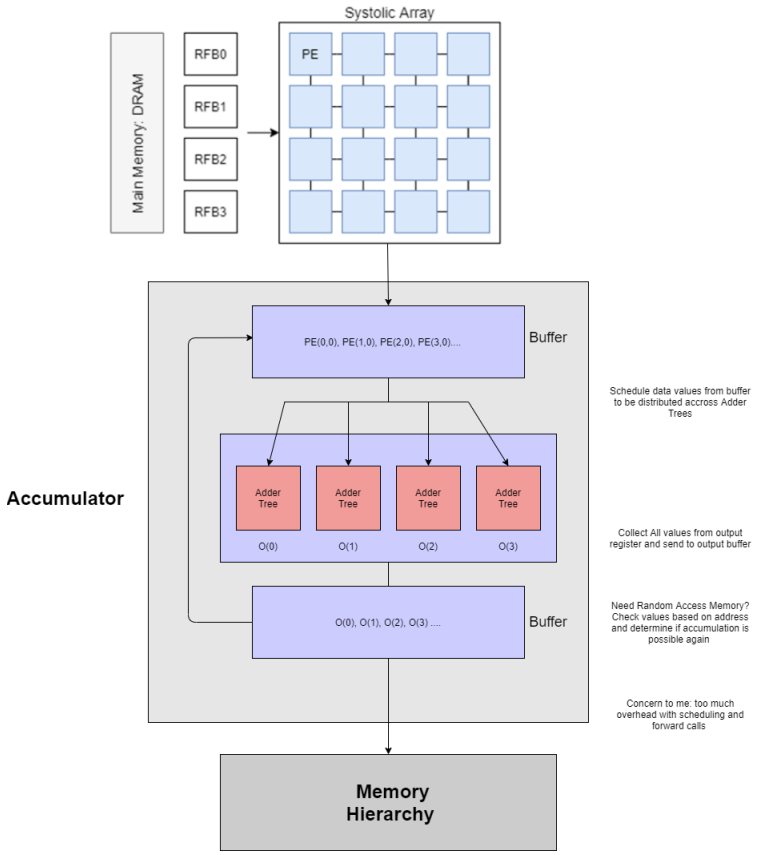

Figure 2.2: Early Stage Accelerator Architecture ©Daniel U. Peralta Velazquez

Final design stages focus on scheduling an prefetching of data as potential solution to adder trees under utilization and hardware size reduction. In this design buffering and registers with padding characteristics were taken into consideration. The final design (Figure 2.3) implements per column buffers and padded registers. The buffer has a multiplier capacity based on the size of input batches of data and its purpose involves allowing to collect multiple batches of data without stalling the Systolic Array or idling the adder trees. The padded register allowed to feed zero values from buffering values into the adder tree and preventing data corruption by padding the rest of the input values.

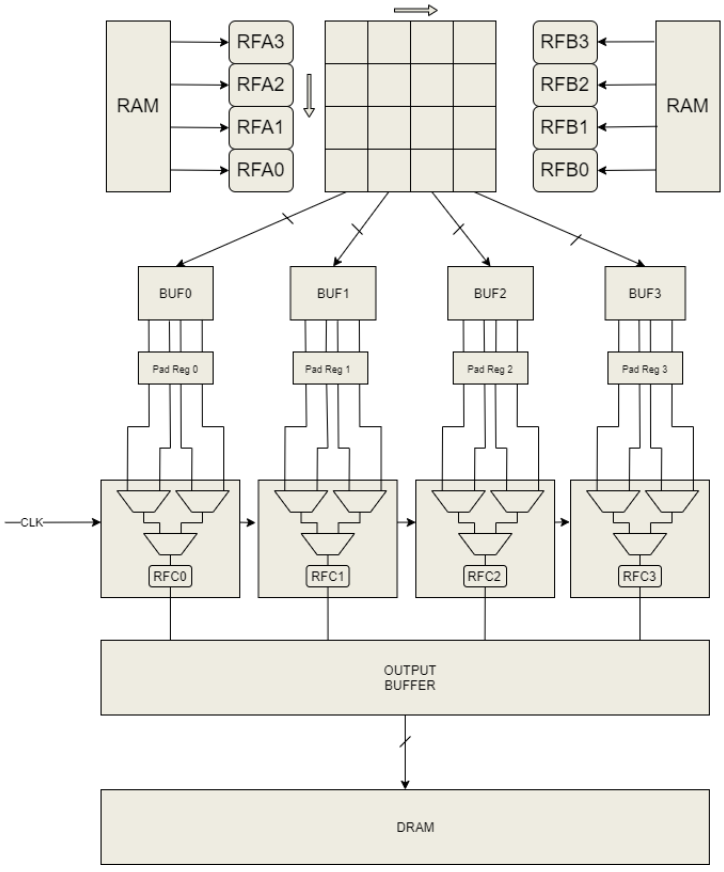

Figure 2.3: Final Proposed Accelerator Architecture ©Daniel U. Peralta Velazquez

2.2 Accumulator: Adder Tree

2.2.1 Adder Tree Building Component

As part of the design of an accumulator microachitecture component an adder tree was implemented to collect computed values from the Systolic Array. The adder trees as shown in Figure 2.4 are composed of interconnected full adders with two inputs and one output value. The connection between these full adders follow a binary tree structure in which the output of a previous level in the adder tree is connected to the inputs of the following level. The data movement is then controlled by series of update and evaluation calls initiated by the main architecture control unit.

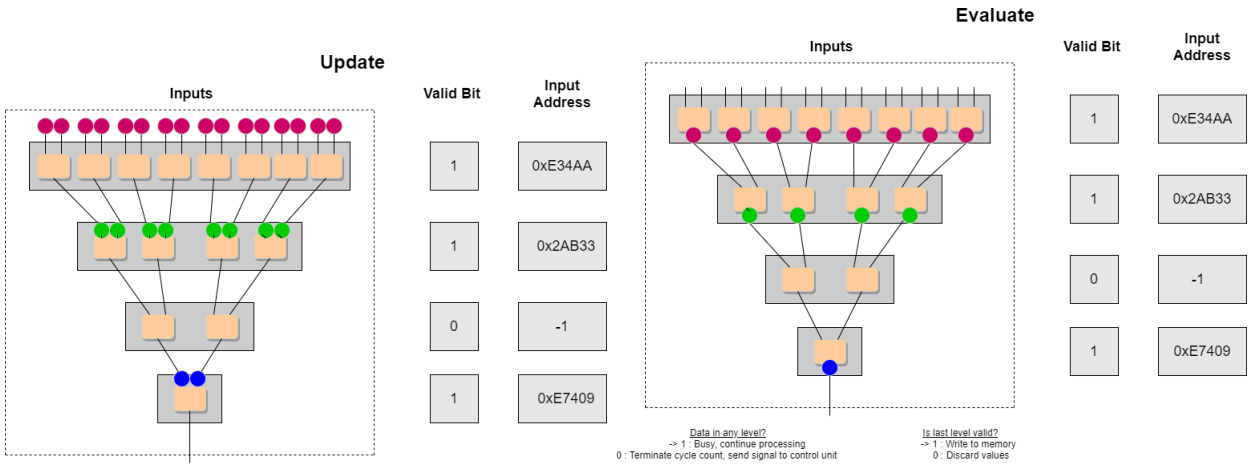

Figure 2.4: Adder Tree: Update & Evaluate Call ©Daniel U. Peralta Velazquez

2.2.2 Scalability

The number of maximum input values depend on the number of rows or columns of PE's in the Systolic array this is to obtain and reduced a one Systolic Array cycle worth of computed values. Figure below shows a representation of an 8-input adder tree suitable for an 8 by 8 Systolic Array. Input values consists of floating point 32 bit.

2.2.3 Pipeline and address forwarding

The Systolic Array will produce batches of data at different amount of clock cycles, therefore, when necessary the adder tree will be stall from new data and a validation flag will maintain track of data belonging to memory. Additionally, the adder tree forward addresses per level to keep track of the destination of reduced values when they reach the output of the accumulator.

2.3 Buffering: Write-out Buffer and Padded register

2.3.1 Write out buffer Design

As part of the collection process a write-out buffer with a first-in-first-out flow method is implemented. As shown in the upper portion of figure 2.5, the buffer receives input batches of systolic array row size through two main wiring components for data values and corresponding addresses. The wiring needed to connect these components increases linearly as the size of the systolic array rows increase.

After directing data and addresses from the systolic array. Data values are inserted one-by-one with their corresponding address. In order to prevent buffer overflow capacity of the buffer is compared to against the size of the incoming data batch to determine if sufficient storage space is available. If sufficient all data points are pushed into the buffer else a flag is raised to allow the control unit to stall the systolic array and preserve the incoming data values until sufficient capacity space is available.

The ejection protocol consists of ejecting computed values into a register. The protocol is defined based on address comparison of address similarity. A pointer to the buffer is then used to select the first-in value address and later checked to determine if adjacent values with equal address are present in the buffer. The comparison method was selected based on possible overhead when comparing addresses. Two options were explored; comparison address across all elements of the buffer and adjacent address checking until a different address was found. Given the data locality from the systolic array the second method is prefer given computed values have address commonality based on dataflow mappings. Additionally, exploring all stored values will increase

overhead significantly as the capacity size of the buffer increases.

2.3.2 Padded Register

A padded register became an essential component to bridge output size from buffer inconsistency and the defined input size of the accumulator component. The padded register purpose consist of prefetching data batches before accumulation by populating the rest of the input batch with zero values. This helps our input value remain consistent in size regardless of the number of output values ejected from the write-in buffer. The size of the padded register as shown in the middle of figure 2.5, consists of a register of accumulator input size as well as a smaller register used to store the corresponding address. After input data has been padded the output register ejects its contents into the input of the accumulator at every evaluate call. The wiring size remains consistent based on the padded register size plus address length and independent from the Systolic Array dimensions. If no valid data is received the register remains zero padded and the address null.

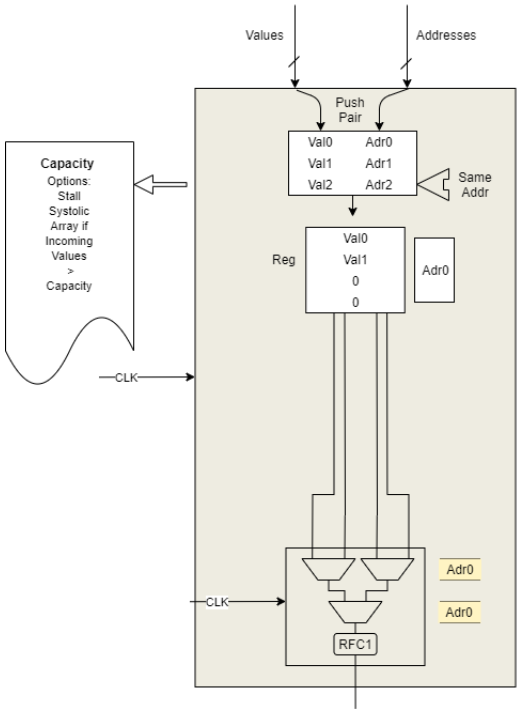

Figure 2.5: Prefetching and scheduling: Buffer & Padded Register ©Daniel U. Peralta Velazquez

2.4 Design and Structure: Systolic Array Simulator (SAS)

In order to test the proposed architectural changes to the accelerator a clock cycle based simulator was designed to simulate the processing of full DNN topologies. The simulator is break down in two main aspects; data generation and collection and structured modulation of architecture components behavior. The later consist of implementing the behavior of the the component through C/C++ classes adhering to structural code as much as possible to facilitate hardware fabrication in future research. Given an architecture component was designed using this code structured several test cases were design under make file options and tests directory structures to test every component for its stand alone behavior. This feature of the simulator allows to verify and test unusual cases that could be present in the overall architecture at some point. These tests are later discuss in this chapter.

The second aspect of the simulator consists of data generation through python scripts and determine by dimensions and quantity of input and filter layers. The topologies of the corresponding DNNs tests is design through a single layer bash file that takes inline inputs to determine the size of every layer. Through this file we can create additional bash files containing interconnected single layers with dimensions corresponding to those of the DNN being tested. The data produced by the architecture is then placed in output maps and can be verify easily based on the properties of the data generated.

2.5 Testing: Systolic Array Simulator (SAS)

Part of the design process consisted of conducting testing on the accumulator components including full adders and adder tree. The phase consisted of designing tests that check for self operation, special cases, data forwarding and validation of the data accumulated. After checking proper behavior of the building components, the next step consisted of validating the adder tree operation and verifying the simulator implementation remained hardware compliant. The following subsections describe testing procedures in detail.

2.5.1 *Full Adder*

The implementation of a single adder consists of two input wires, two inputs, and one output. Additionally, the adder has an “Update” and “Evaluate” function call. Update is takes data on the wires of the adder an initializes them as inputs. Evaluate performs a summation of both input values and assigns the result to the output component. In order to check for proper functionality of the adder, a test was created in which the correct sequence of functions was (update -> evaluate Test 1) was implemented and other test with the wrong order of function calls was executed (update->update->evaluate – Test 2) Figure 2.6. This help us understand the behavior of the adder each clock cycle and maintain a structural code approach for easier hardware application.

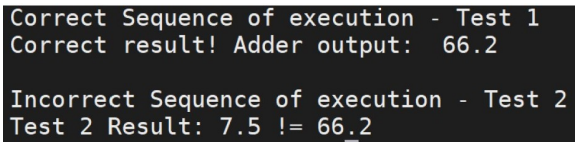

Figure 2.6: Adder Test Results ©Daniel U. Peralta Velazquez

2.5.2 *Adder Tree*

The formation of every adder tree was derived from the adder testing and design. Every adder tree within simulation can be instantiated with a given input size and further levels are created to reach a reduction to one. Once every adder tree is created in simulation, a sequence of "evaluation" and "update" function calls is executed to reduce and move all values in every level in the adder tree. The simulator is able to handle input data with no real destination address or meaningful value these data is also known as stall cycles. Additionally, the testing and simulation of the adder tree maintains track of validation bits, output register, and terminates the adder tree operation upon return of a false busy flag from the evaluate function call. Additional test options are shown in Figure 2.7 .

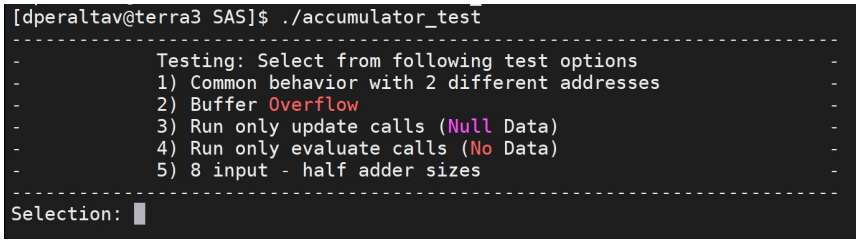

Figure 2.7: Available Subsystem Integration Tests ©Daniel U. Peralta Velazquez

2.5.3 Write-out Buffer

In order to properly simulate the behavior of a buffer component a instance of deque from the standard library is used. The deque is capable of pushing a pair value of type value-address and is controlled based on a predefined capacity set by the user at compilation time. The deque was tested for overflow by fetching batches of data with singular addresses to increase the input flow and decrease output flow as much as possible. This test allow us to reach limit capacity of the buffer and determine if invalid access or behavior is preventable. The test allow us to define the number of clock cycles to overflow the buffer as well as perceive the state of other connected components.

2.5.4 Padded Register

The padded register holds an important role when prefetching and scheduling data between the buffering component and accumulator. The testing of the padded register (Figure 2.7) consisted of collecting ejected values from buffer and determining if the new batch of data was properly adjusted in size in reference to the adder tree input. This resizing consisted of folding larger inputs of data, zero padding smaller inputs of data and properly feeding the output batch of data into the corresponding full adder level of the adder tree.

2.5.5 Adder Tree size reduction

The integration of a buffering system, padded register, and the overall adder tree accumulators has the intend to reduce hardware while maintaining a similar accumulation flow. In order to

test for hardware reduction particularly adder tree input size, test were conducted were the size of the adders were reduced by half its size and executed for as many clock cycles as the user prefer. As shown in Figure 2.8, the test consisted of introducing two different batches of data with common address to determine if the buffering component was folding correctly and the accumulator utilization increase. This test can be modify to include larger and different batches of input data along with a variability of execution of computing clock cycles.

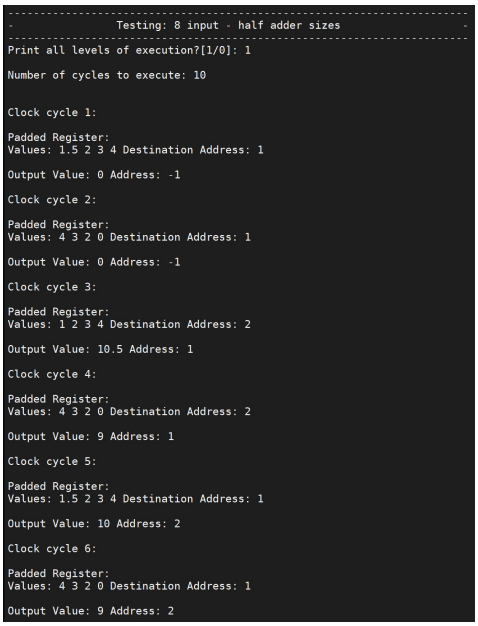

Figure 2.8: 8 input test with half adder tree sizes ©Daniel U. Peralta Velazquez

3. RESULTS

Given previous testing of the functionality and reliability of every component that forms the accumulation stage of the accelerator the integration of such laid a great reduction of computing clock cycles and memory accesses.

In order to asses the improvement of the accumulator based on the per-column adder tree components a full execution of CNN layers was conducted.

3.1 AlexNet CNN Execution using SAS and Accumulation

The execution of the convolution of CNN layers was conducted with the use of the HPRC cluster. The simulation consisted of modifying an input bash file to define the dimensions and properties of the CNN to be executed. As shown in Figure 3.1, we define the dataflow as "CW2" referring to channel-wise as well as the number of filters and dimensions of every tensor. After defining every parameter, data is generated and the execution of the data is conducted. The time of every simulation is proportional to the number of layers and size of filters. Additionally, the clock execution simulator is able to detect mismatch errors used to determine the reliability of the computed data.

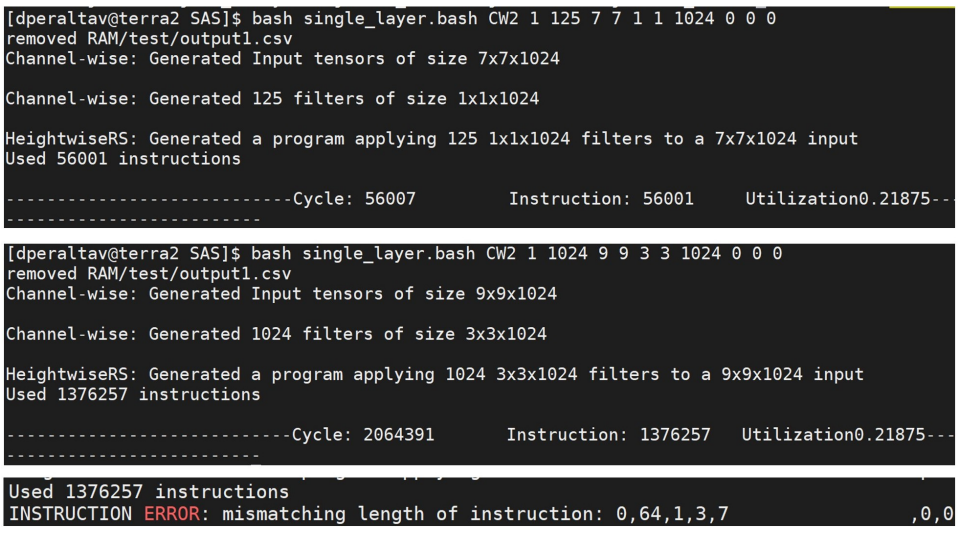

Figure 3.1: CNN Layer Execution & Error Handling ©Daniel U. Peralta Velazquez

As shown in Figure 3.2, The implementation of an accumulator produced an incremental improvement of 2.15x on runtime in AlexNet executions. The results were obtained using a 32-by-64 Systolic Array. The utilization of the Systolic Array in both implementation remain the same despite the difference in clock cycles. This can be attribute to the multiple reductions of computed values and reduction of execution time spent in memory handling.

Alexnet

systolic array 32x64 Using Channelwise dataflow

Layer	single cycle cyles	utilization	cycles pilined	utilization
1	22,146,433	4.48%	22,146,439	4.48%
2	85,487,361	67.99%	29,102,086	67.99%
3	1,368,577	34.38%	304,134	34.38%
4	2,052,865	34.38%	456,198	34.38%
5	1,368,577	34.38%	304,134	34.38%

Figure 3.2: AlexNet CNN Layer Execution & Cycle Count ©Daniel U. Peralta Velazquez

3.2 YoloTiny CNN Execution using SAS and Accumulation

Additional execution of full DNNs was conducted. As shown in figure 3.3, YOlOTINY produced an improvement of 3.3x on runtime executions using a 32-by-64 Systolic Array. These improvement results are based on clock cycle count alone. The runtime improvement shows a positive increase of performance compared to an increase of the amount and size of interconnected layers. In order to test this hypothesis other complex DNN models can be use to compute a larger number of layers and data.

These results remain consistent when introducing buffering and padding to increase the accumulator utilization and reduce its hardware size. The equal utilization of the Systolic Array with and without an accumulator component remains the same. This metric along output data and instruction/data error handling in the architecture designed files allow to verify the results of the

computation run.

YOLOTINY

systolic array 32x64 Using Channelwise dataflow

Layer	single cycle cyles	utilization	cycles pilined	utilization
1	1,550,017.00	4.66%	1,550,023.00	4.66%
2	1,515,361.00	22.88%	826,567.00	22.88%
3	1,241,089.00	39.45%	465,415.00	39.45%
4	1,036,801.00	78.13%	230,407.00	78.13%
5	995,329.00	75.00%	221,191.00	75.00%
6	1,824,769.00	34.38%	405,511.00	34.38%
7	5,971,969	28.13%	1,327,111.00	28.13%
8	9,289,729.00	21.88%	2,064,391.00	21.88%
9	350,001.00	21.88%	56,007.00	21.88%

Figure 3.3: YoloTiny CNN Layer Execution & Cycle Count ©Daniel U. Peralta Velazquez

3.3 Metrics

In order to test that these system level architecture and microarchitecture design changes decrease memory accesses and latency, three metrics need to be measured. The first metric is utilization. Given the size of layers and computational unit remain the same the utilization of the Systolic array should be similar in all design changes. This metric will serve as a validation step when validating a full batch of processing data. The second metric is cycle count. In the simulation model writes and reads from memory take one clock cycle and data is prefetched to the Systolic Array before cycle count begins. This approach aims to emphasize the overall parallelism achievable by different dataflows and allows to quantify the number of memory accesses. The third metric is cache size. The simulator is design to provide memory accesses patterns based on selected dataflow. In order to determine the optimal size for internal caches and buffers these memory accesses patterns are analyze to find the appropriate capacity for storage components.

Data correctness verification: As part of the simulation process the most important step when designing and integrating new architecture changes becomes the correctness and validation of the output data. The simulator produces an output memory map used to verify the throughput of every dataflow and the correctness of architectural changes. When generating the output memory map. array utilization and memory accesses per cycle are logged and help determine the impact of new design changes. In order to make the verification of computed data feasible deterministic data generator and averaging filters were applied to easily compute and verify the output of a layer based on the filter and layer size properties.

4. CONCLUSION

4.1 Architecture Design

Throughout this research project the main goal consisted of designing hardware architecture components capable of addressing memory bottlenecks with current accelerator architectures dataflows. The proposed hardware components had an extensive design process in which system-level and microarchitecture was explored. The design components have features that facilitate its integration and overall reduction of computing cycles. The main features are described below.

4.1.1 Architecture Design: Accumulator

The accumulator component is the main component when reducing cycle counts and memory accesses. The overall success of this component derives from an optimal reduction of number rows amount of computed values to one. This component is defined a multiplier of the size of the Systolic Array and directly mapped to external memory hierarchies. The accumulator component allows for reductions to occur as close as possible to the main computing unit and allows to avoid in memory operations to further reduce the dependency on off chip memory.

4.1.2 Architecture Design: Buffering & Padded Register

The buffering and padded register components serve as the main components when prefetching and maintaining data consistency when computed values are ejected from the Systolic Array. These components play an important role in the scalability of adder trees and integrity of data when receiving batches of data from the Systolic Array. The accumulator component allows for input consistency as well as an independent hardware input size of the accumulator that directly translates to a reduction in hardware while maintaining performance in comparison with direct accumulator and Systolic Array connection.

The overall integration and design of proposed architectural changes was successful. The overall components allowed for a significant reduction of clock cycles, consistent memory pat-

terns, and higher hardware utilization in comparison to previous accelerator architectures based on a Systolic Array.

4.2 Future work and areas to explore

This study introduces a Scalable Systolic Array Architecture with effective reduction and scheduling components that allows for the exploration of different microarchitectural changes based on dataflow mappings. This framework allows for the exploration of different sizes and amount of interconnected layers that allows to design and generate data for different DNN models. Additionally, we are able to explore hardware sizes for all components in the architecture as well as exploring parallelism techniques while reducing the memory footprint. These attributes in the Systolic Array Simulator open the doors to future exploration of data mappings, hardware size, and DNN model exploration.

Potential future work in this study includes computing different and current DNN models to obtain the performance of the accelerator architecture with more complex and deeper models. Not only is the performance of the accelerator based on the DNN models. There are many accelerator architectures that scale at an alarming rate with heavy data movement, in order to understand the impact of hardware size a comparison between other systems such as a the Tensor Processing Unit from Google will help determine the overall size to ratio performance of our proposed architecture and offer a possibility to processing on edge devices. Additionally, once validation of the architecture is complete and physical hardware compliance is fully met fabrication steps will follow to determine the fabrication of the accelerator at an optimal physical space. The physical validation will consist of several component timing analysis, component mapping, appropriate wiring and overall integration of the system.

REFERENCES

[1] A. Roy, J. Sun, R. Mahoney, L. Alonzi, S. Adams, and P. Beling, "Deep learning detecting fraud in credit card transactions," in *2018 Systems and Information Engineering Design Symposium (SIEDS)*, pp. 129–134, 2018.

[2] S. Samsi, C. J. Mattioli, and M. S. Veillette, "Distributed deep learning for precipitation nowcasting," 2019.

[3] W. Zhu, L. Xie, J. Han, and X. Guo, "The application of deep learning in cancer prognosis prediction," *Cancers*, vol. 12, p. 603, Mar 2020.

[4] C. Lopez Carrasco, "Channel-wise row stationary. a systolic array dataflow for accelerating convolutional neural networks," Sep 2020.

[5] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "Scale-sim: Systolic cnn accelerator simulator," 2018.

[6] A. S. Hyoukjun Kwon and T. Krishna, "Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2017.

[7] F. Rosenblatt, *Principles of neurodynamics: perceptrons and the theory of brain mechanisms*. Report (Cornell Aeronautical Laboratory), Spartan Books, 1962.

[8] 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.), pp. 1097–1105, Curran Associates, Inc., 2012.

[9] Kung, "Why systolic architectures?," *Computer*, vol. 15, no. 1, p. 37–46, 1982.