5-8-2019

Approximate Computing in Coarse Grained Reconfigurable Architecture

Lincoln Douglas Kinley
Southern Illinois University Carbondale, lincoln.kinley@siu.edu

I would like to thank Dr. Iraklis Anagnostopoulos and Ioannis Galanis for their support in developing my researching abilities and helping me with this undergraduate research project. I would also like to thank the SIUC Honors Program for their guidance and mentorship through my bachelor's degree. Finally, I would like to thank my friends and family for supporting me throughout my studies.

Approximate Computing in Coarse Grained Reconfigurable Architecture

Lincoln Kinley

A thesis submitted to the University Honors Program

In partial fulfillment of the requirements for the

Honors Certificate with Thesis

Southern Illinois University

May 8, 2019

Abstract

Approximate computing has emerged as a new computing paradigm capable of reducing the power requirements for or accelerating some workloads. Due to cascading error and the nature of binary arithmetic, it is difficult to predict the exact effects that approximation may have on an error tolerant workload. In this work, we implemented configurable levels of approximation into a Coarse Grained Reconfigurable Architecture (CGRA) to study the effects of error tolerant algorithms on an approximate CGRA. We will use the CGRA Compilation Framework which simulates a CGRA using gem5, and we will implement the approximate hardware using multiple different approximate arithmetic modules included in Low Power Approximate Computing Library. Finally, we will perform a hardware level simulation on approximate modules to estimate the reduction in power from using approximate hardware.

Acknowledgements

I would like to thank Dr. Iraklis Anagnostopoulos and Ioannis Galanis for their support in developing my researching abilities and helping me with this undergraduate research project. I would also like to thank the SIUC Honors Program for their guidance and mentorship through my bachelor's degree. Finally, I would like to thank my friends and family for supporting me throughout my studies.

Table of Contents

Table of Figures

Table of Tables

Introduction

Ever since we hit the power wall, advancements in computing have been restricted by three main factors. Specifically, hardware can have two of the following three: performance, efficiency, and generality. In order to achieve all three of these, a new computing paradigm, approximate computing, has emerged. Approximate computing introduces a fourth dimension, accuracy, thus expanding the level of control programmers have. By reducing the requirements for data to be completely accurate, energy efficiency can be increased, while also accelerating algorithms [1].

Approximate computing is a very wide field, with solutions that exist at the software and hardware level. Approximate computing is mostly geared towards accelerating data heavy algorithms are not required to be fully accurate in order to function. This includes algorithms that fall under recognition, mining, and synthesis (RMS) applications. Although approximate computing can be used to accelerate many algorithms, it is not a definite solution for developing faster and more efficient algorithms, and as such it will likely never replace traditional exact computing. Instead, approximate computing allows the acceleration of a program [2]. There are multiple different ways to implement approximate computing, which includes solutions at the software level, and the hardware level. At the software level, iterative algorithms can reduce their iterations, thus reducing the accuracy and saving compute power. At the hardware level, arithmetic functions can drop some of the low significance bits to speed up their frequency or reduce their power consumption [2].

Coarse Grained Reconfigurable Architectures (CGRAs) are programmable computer hardware devices that use a large number of functional units which share data with adjacent functional units very quickly. Each of these functional units has all the aspects of a traditional information processor, including an Arithmetic Logic Unit (ALU) and registers.

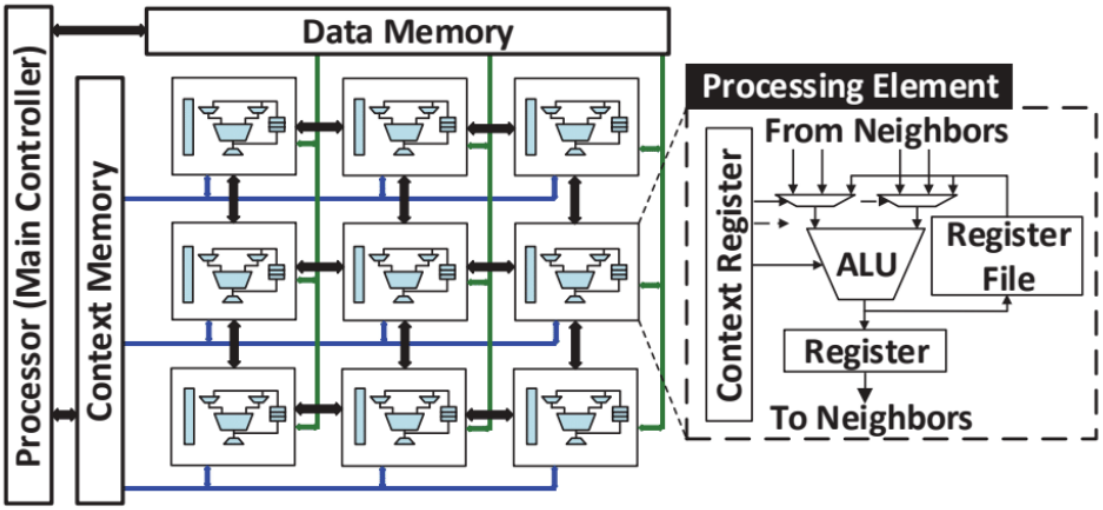

Figure 1: CGRA Architecture [3]

This project has the goal of allowing the programmer to choose an approximation tolerance of his or her application. The application can then accelerate data processing by using the approximate CGRA to meet the tolerable approximation level, as previous work in approximate computing has shown that even marginal decreases in accuracy can have significant impact on efficiency and speed at which the software runs [3 4].

We accomplish this by dedicating functional units in the CGRA to process data at a specific accuracy level and processing the data on functional units of varying accuracy. Data processing can reach a configurable accuracy level stated by the programmer by using a specific number of each approximation level functional unit.

Motivation

Research into Approximate CGRAs has two primary motivation. The first is to develop a low powered, general processor, and the second is to devise a method to accelerate error tolerant algorithms. An approximate CGRA has the potential to accomplish either of these, the choice would ultimately be up to a hardware designer to design the device to fit a power profile.

The Internet of Things (IoT) is a growing system, which utilizes many different low power distributed computers. Due to the remoteness of the device, many IoT devices run off of battery power. The Approximate CGRA aims to reduce the power requirements of processing data, allowing IoT devices powered by a battery to be powered for a significantly longer time off a single charge. This increases the usefulness of the device, while still being capable of retaining primary functionality. Furthermore, many IoT devices utilize sensors, which are already error tolerant, meaning that approximate computing can easily be integrated into their functionality.

Another motivation of the approximate CGRA is its ability to accelerate error tolerant algorithms in many systems. In particular, we are considering algorithms that fall into the RMS category. Approximate computing has already been shown to increase the data throughput of error tolerant algorithms [4].

It is important to understand that this project does not seek to replace exact computing. There are numerous tasks and algorithms that cannot be approximated, as approximation would result in crashes or otherwise errant behavior. Exact computing will always remain an important aspect of computing, while approximate computing seeks to supplement exact computing by accelerating error tolerant algorithms.

Methods

Our method involved implementing approximate hardware into the functional unit of a CGRA. This included the adder and multiplier circuit inside the ALU of each functional unit. We also allowed both the adder and multiplier circuit inside the ALU to be configured to approximate a configurable number of bits, allowing for a high level of control when utilizing approximate computing.

Tools

We used a number of existing tools to implement our design, both for the design of approximate computing modules and the simulation of the CGRA.

Low Power Approximate Computing Library (lpAClib) from the Chair for Embedded Systems in the Karlsruhe Institute of Technology was used to implement our approximate computing modules. lpAClib includes a total of three different approximate adders, and two different approximate multipliers. lpAClib allows for the users to specify the total number of bits to be approximated, which allows for more control over the level of approximation of the output. This proved to be very useful for achieving an optimal level of approximation in an algorithm. lpAClib also allows for the multiplier to use any of the implemented adders. This feature could be useful in the future to create an adder that has a high precision, and a multiplier that has a low precision, or vice versa. Given the application dependent results of using approximate computing, this feature is highly desirable. While we were working to implement approximate computing with the CGRA, we decided to rewrite the lpAClib library using the bitset type instead of the char type. We did this for two reasons. First of all, we found that lpAClib has a memory leak, which we were not able to identify, and second of all, using bitsets makes the problem use far less memory than the original implementation of lpAClib. Our implementation of lpAClib requires using a C++ compiler, whereas the original lpAClib is compatible with C [5].

For the simulation of the CGRA, we used the CGRA Compilation Framework (CCF) by Shail Dave in the Compiler Microarchitecture Lab at Arizona State University. CCF is built on gem5, an implementation level simulator. Gem5 is an open source system level simulator designed to simulate the functionality of hardware, however it does not simulate specific details like power and area. This allows us to rapidly test the outputs of many different configurations of the system,

however we are limited to the amount of data we can get out. Our output has the exact output that an approximate CGRA would have, however we do not have a proper power, timing, and area analysis of the CGRA [6].

Results

We tested a total of two different algorithms using the approximate CGRA, both being actual applications. Applications tend to act differently than simple arithmetic, as particular operations will be favored over others, meaning some applications will see great improvement with few losses using approximation, while other applications may see little improvement with great losses. The use of applications gives a greater perspective on real results. The below summary of approximate arithmetic modules has highlighted functions to indicate that the output does not match its corresponding accurate module.

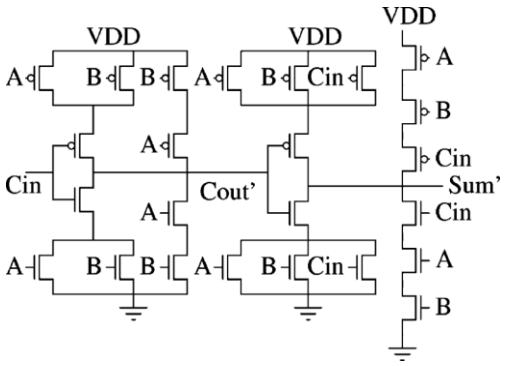

Figure 2: Accurate Adder Circuit [7]

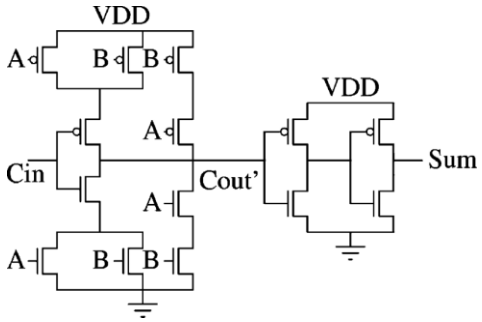

Figure 3: Impact Zero Approximate Adder Circuit [7]

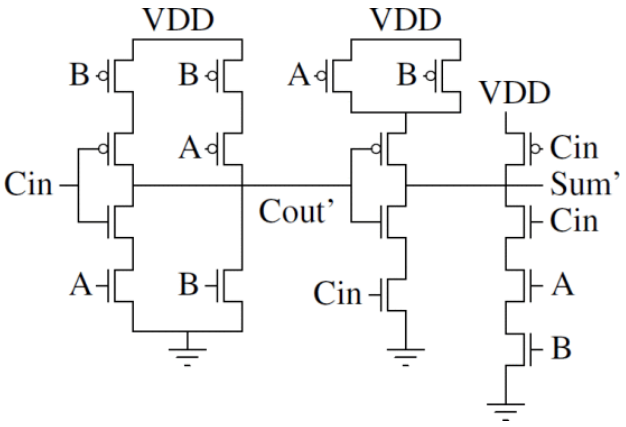

Figure 4: Impact First Approximate Adder Circuit [8]

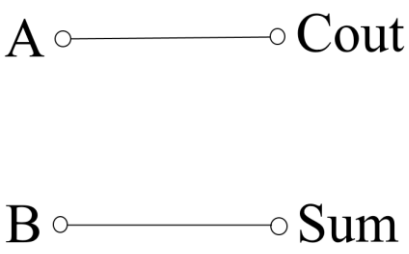

Figure 5: Impact Third Approximate Adder Circuit [8]

Accurate Add				
A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 1: Accurate Adder Truth Table

Impact Zero Approximate Add				
A	B	Cin	Sum	Cout
0	0	0	1	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	0	1

Table 2: Impact Zero Approximate Adder Truth Table

Impact First Approximate Add				
A	B	Cin	Sum	Cout
0	0	0	1	0
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	0	1

Table 3: Impact First Approximate Adder Truth Table

Impact Third Approximate Add				
A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	0	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1

Table 4: Impact Third Approximate Adder Truth Table

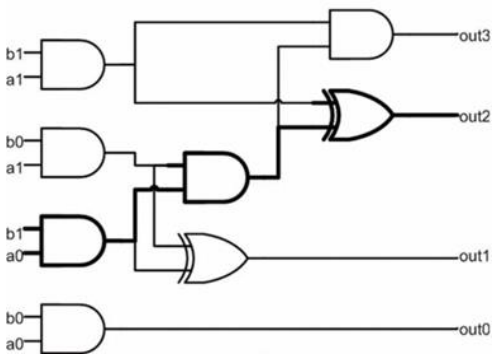

Figure 6: Accurate Multiplier Circuit [9]

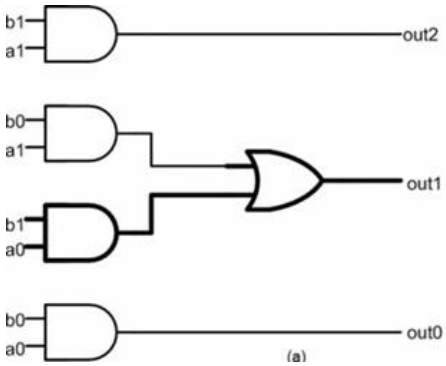

Figure 7: Impact Zero Approximate Multiplier [9]

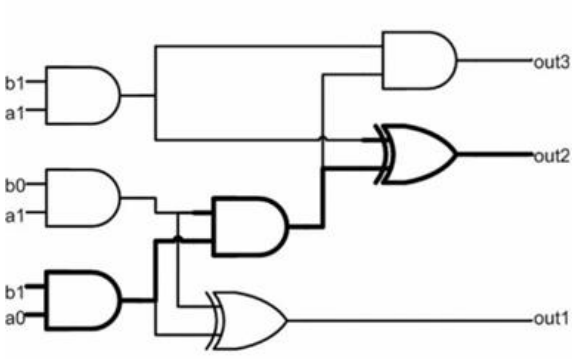

Figure 8: Impact First Approximate Multiplier [5 9]

Exact Multiply				
	00	01	10	11
00	0000	0000	0000	0000
01	0000	0001	0010	0011
10	0000	0010	0100	0110
11	0000	0011	0110	1001

Table 5: Exact Multiplier Truth Table

Impact Zero Approximate Multiply				
	00	01	10	11
00	0000	0000	0000	0000
01	0000	0001	0010	0011
10	0000	0010	0100	0110
11	0000	0011	0110	0111

Table 6: Impact Zero Approximate Multiplier Truth Table

Impact First Approximate Multiply				
	00	01	10	11
00	0000	0000	0000	0000
01	0000	0000	0010	0010
10	0000	0010	0100	0110
11	0000	0010	0110	1000

Table 7: Impact First Approximate Multiplier Truth Table

The first application we tested using the CGRA was the isqrt function included in the MiBench benchmarking tool. The isqrt function computes the square root of the input, using almost entirely bit shifting operators, and exclusively integer arithmetic. This makes the isqrt function fast on a wide array of systems, and easy to implement on the approximate CGRA. We tested the isqrt function with all three approximate adders at four, eight, twelve, and sixteen bits approximated. We compared the effects on percent error that the number of bits approximated has for each adder, as well as a comparison of the percent error of each adder.

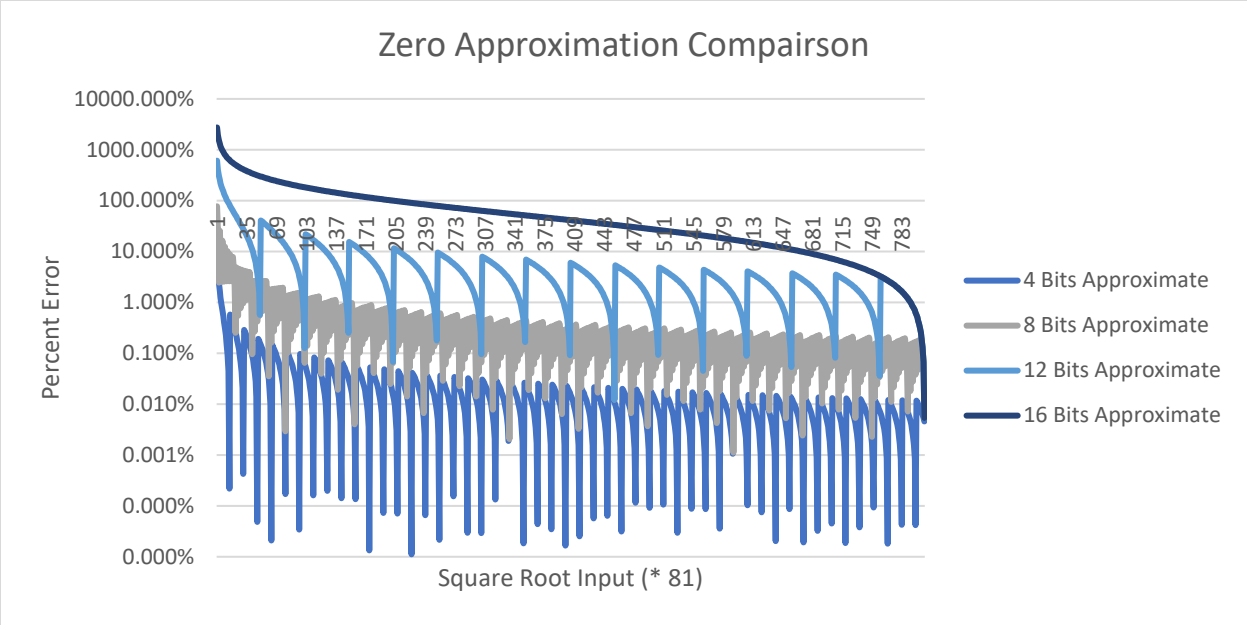

Figure 9: Zero Approximation Comparison

In this application, the percent error of the adder follows a pattern of decreasing before suddenly spiking. Increasing the number of bits approximated reduces the average error, as well as increases the frequency that the percent error follows this pattern.

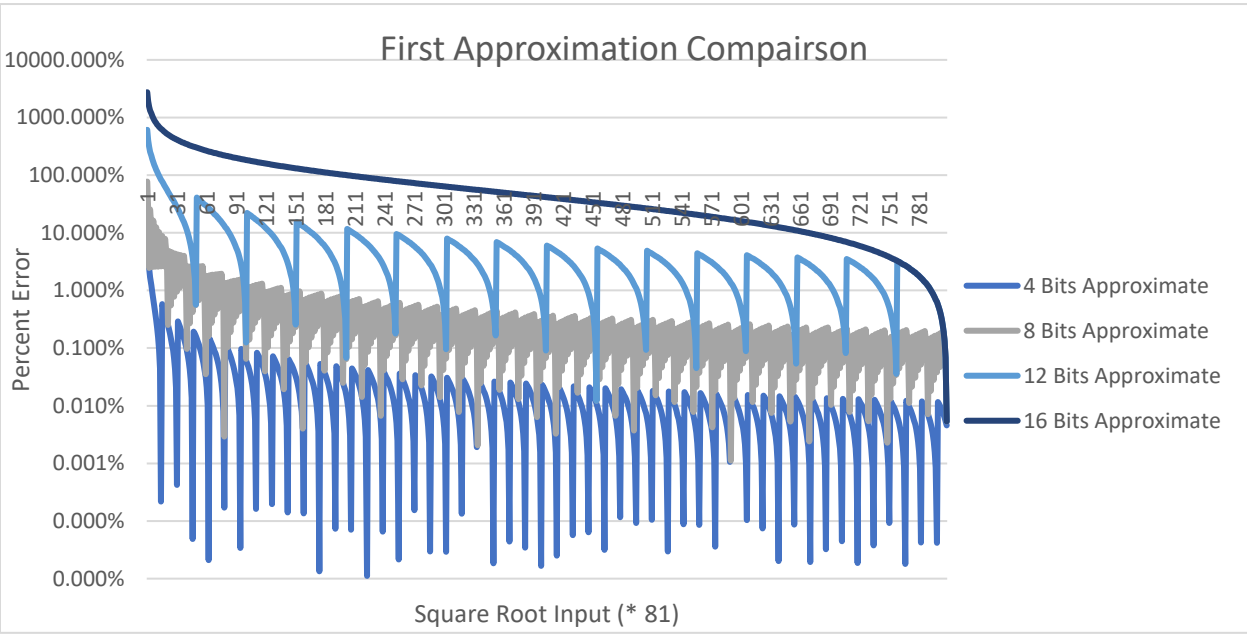

Figure 10: First Approximation Comparison

The output of the Impact First Approximation was found to be exactly the same as the Impact Zero Approximation. This becomes more apparent when comparing Impact Zero and Impact First Approximate adders directly to each other. As Impact First Approximate adder functions the same as Impact Zero Approximate adder, it is likely that this application never has an addition where A is zero, B is one, and Cin is zero, as this is the only difference case between Impact Zero and Impact First approximate adders.

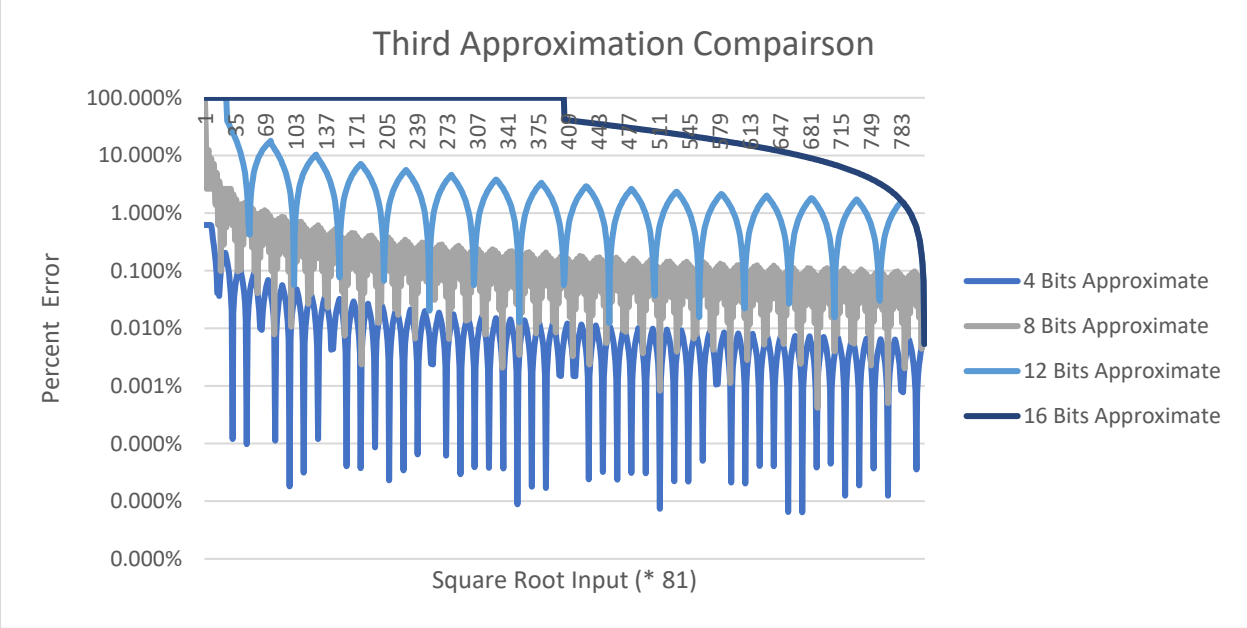

Figure 11: Third Approximation Comparison

Impact Third Approximate adder follows an almost identical pattern as Impact Zero and Impact First Approximate adders. The difference here is the percent error gradually increases to a peak, instead of spiking. The second half of the pattern is exactly the same as the Impact Zero and Impact First Approximate adders. The frequency of the pattern is also the same as the Impact Zero and Impact First Approximate Adders

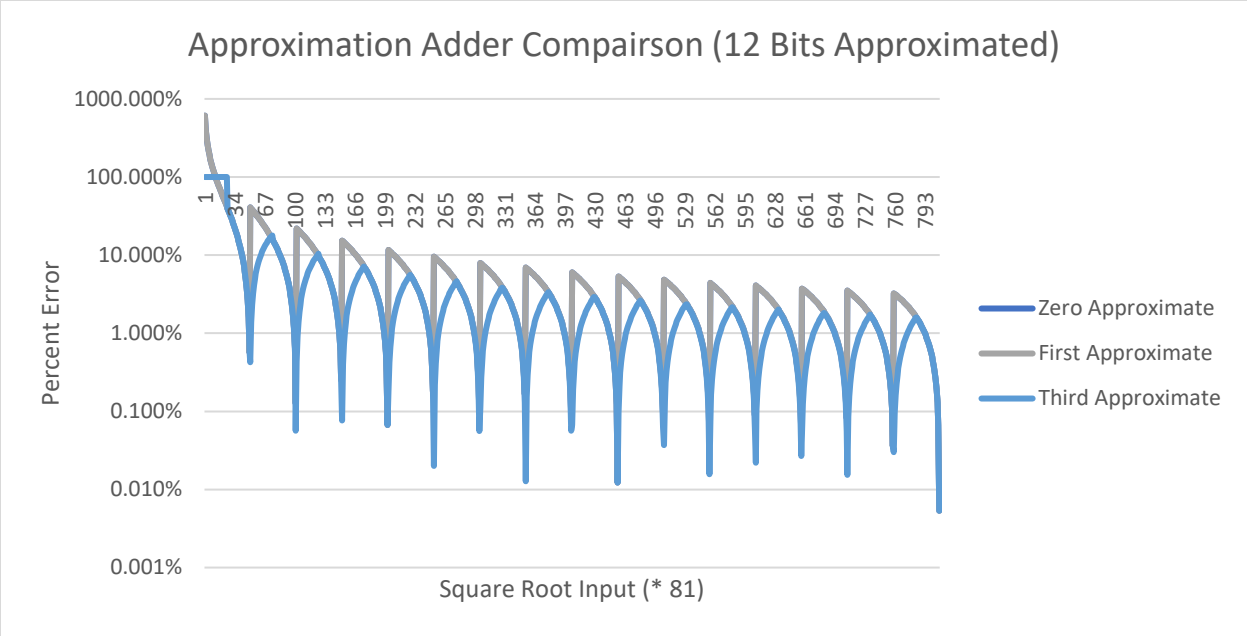

Figure 12: Approximate Adder Comparison

By comparing the percent error from each approximate adder, we can find which adder performed best at this application. We find that Impact Third Approximation performed equally or better than the other two approximate adders, with the exception of the second quarter of the first iteration of the repeating pattern, in this case between 1053 and 2025.

The second application we tested was a Sobel edge detection algorithm on a test image. The Sobel edge detection functions on a greyscale image and detects areas that rapidly change from a dark shade to a light shade. This application is the most similar of the two that we tested to a real application that would be used in most scenarios. The issue with the Sobel edge detection is that the results are not quantifiable, meaning we cannot quantify how close the approximate solution is to the exact solution. In this case, we used an Impact Third Approximate Adder and an Impact First Approximate Multiple and approximated the 16 least significant bits.

Figure 13: Original Image (left) and Scaled Down Image (right)

Figure 14: Exact Blur Image (left) and Exact Peaks Image (right)

Figure 15: Approximate Blur Image (left) and Approximate Peaks Image (right)

We also did a transistor level simulation of the Approximate Adder circuits to determine some of the physical characteristics of an approximate computing module. vhdl code for the implementation of each approximate and exact arithmetic adder and multiple is included in lpAClib. These modules were simulated to find the area the circuit would take up on an integrated circuit. We also counted the transistors required to implement an approximate module. We use the number of transistors to estimate the power reduction for the approximate modules.

		One Bit	Percent Reduction (vs. exact)	16 Bit Accurate 16 Bit Approximated	Percent Reduction (vs. exact)
Exact Adder	Area (um)	40.5	0 %	1784.5	0 %
	Transistors	24	0 %	768	0 %
Impact Zero Approximate Adder	Area (um)	24.8	38.8 %	1532.3	14.1 %
	Transistors	14	41.6 %	608	20.8 %
Impact First Approximate Adder	Area (um)	16.2	60 %	1392.5	22.0 %
	Transistors	16	33.3 %	640	16.7 %
Impact Third Approximate Adder	Area (um)	0.2	99.5 %	1132.3	36.6 %
	Transistors	0	100 %	384	50 %

Table 8: Power and Area

Steps for Replication

In order to replicate our work, modifications must be made to the CGRA Compilation Framework. When gem5 is compiled, it compiles using a static CGRA x dimension and y dimension. This means the definition of the x and y dimensions must be changed and gem5 recompiled to simulate different sizes of CGRA. The x and y dimensions of the CGRA are defined in multiple places. All of these definitions must be changed to matching values in order for gem5 to run properly.

There are also cases where the file path of certain files is hardcoded. In some cases, the file path can be left as is with reduced functionality, but in other cases the file path must be changed to function properly.

In order to accelerate the process of changing the x and y dimension of the CGRA as well as fixing the file paths, we created a python script that handles all the modification automatically. This python script can be found in our private repository, which can be accessed by contacting Dr. Iraklis Anagnostopoulis.

Conclusions

We can draw two different conclusions from this research project. We found that an algorithms actual percent error may vary from the percent error of the approximate arithmetic used, and we found that using approximate computing modules can reduce the size of a chip, and number of transistors to implement the circuit on a chip.

The isqrt function from MiBench showed that for most possible inputs, the Impact Third Approximate adder performed equal to or better than the other two adders when using the same number of bits approximated. The Impact Third Approximate adder is the most approximate adder, showing that the accuracy of the adder used does not always correlate to the accuracy of the output of an algorithm. Instead, some algorithms may benefit more from using a particular type of approximate hardware.

By comparing each approximate hardware circuit, we can see that using approximate hardware decreases both size of the circuit as well as transistors required to implement the circuit. Both imply that the approximate hardware uses less power than an accurate module. In turn, the operating frequency could be increased to maintain a similar power profile as accurate arithmetic units, but with increased processing power.

References

[1] A. Lotfi, A. et al. "Grater: An Approximation Workflow for Exploiting Data-Level Parallelism in FPGA Acceleration," *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, 2016, pp. 1279-1284.

[2] J. Han and M. Orshansky, "Approximate computing: An Emerging Paradigm for Energy-Efficient Design," *2013 18th IEEE European Test Symposium (ETS)*, Avignon, 2013, pp. 1-6.

[3] O. Akbari, M. Kamal, A. Afzali-Kusha, M. Pedram and M. Shafique, "PX-CGRA: Polymorphic approximate coarse-grained reconfigurable architecture," *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, 2018, pp. 413-418.

[4] Brandalero, M. et al. "Approximate On-The-Fly Coarse-Grained Reconfigurable Acceleration for General-Purpose Applications," *2018 55th Design Automation Conference (DAC),* San Francisco, 2018

[5] Muhammad Shafique, Rehan Hafiz, Semeen Rehman, Walaa El-Harouni, Jörg Henkel, "A Low Latency Generic Accuracy Configurable Adder", in 53nd ACM/EDAC/IEEE Design Automation Conference & Exhibition (DAC), 2016.

[6] Dave, Shail, Mahesh Balasubramanian, and Aviral Shrivastava. "RAMP: resource-aware mapping for CGRAs." 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC). IEEE, 2018.

[7] V. Gupta, D. Mohapatra, A. Raghunathan and K. Roy, "Low-Power Digital Signal Processing Using Approximate Adders," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 1, pp. 124-137, Jan. 2013.

[8] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan and K. Roy, "IMPACT: IMPrecise adders for low-power approximate computing," *IEEE/ACM International Symposium on Low Power Electronics and Design*, Fukuoka, 2011, pp. 409-414.

[9] P. Kulkarni, P. Gupta and M. Ercegovac, "Trading Accuracy for Power with an Underdesigned Multiplier Architecture," *2011 24th Internatioal Conference on VLSI Design*, Chennai, 2011, pp. 346-351.