Approximate Computing in Coarse Grained Reconfigurable Architecture

Lincoln Kinley, Ioannis Galanis†, Iraklis Anagnostopoulos†

†Southern Illinois University at Carbondale, Department of Electrical and Computer Engineering, Carbondale, IL. 62901

Abstract

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 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 that 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 algorithms, it is not a definite solution for developing faster and more efficient algorithms, and as such it will likely never replace traditional computing.

There are multiple different ways to implement approximate computing. 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 buts or adjust the logic to reduce the number of logic gates to increase the computational power or reduce the power consumption [2].

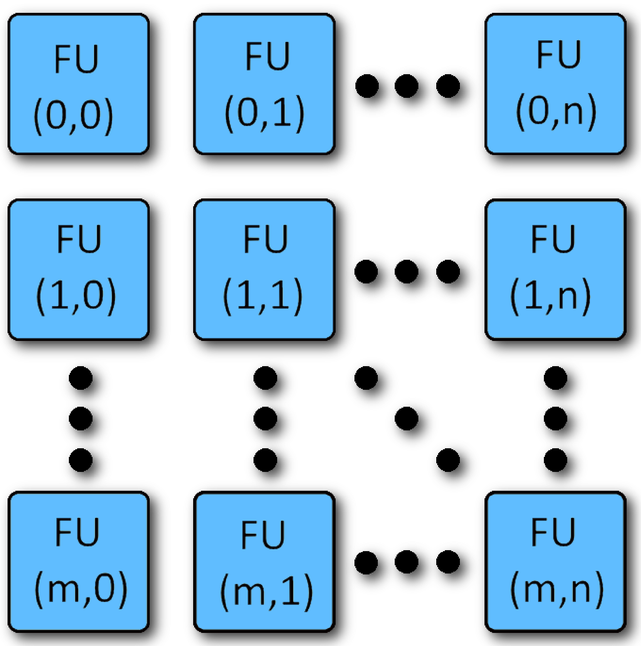

Figure 1: A n by m CGRA implementation

Introduction

Coarse Grained Reconfigurable Architectures (CGRAs) are programmable computer hardware 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 CPU, including an ALU and registers.

This project has the goal of allowing the programmer to choose a 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].

We accomplish this by dedicating functional units in the CGRA to process data at a particular 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.

Operation	Result	Actual	Error	Percent Error
13 × 115	5647	1495	4152	277.73
25 × 92	9567	2300	7267	315.96
59 × 162	14335	9558	4777	49.98
102 × 92	14399	9384	5015	53.44
153 × 120	23103	18360	4743	25.83
198 × 223	40783	44154	-3371	7.63
228 × 244	51455	55632	-4177	7.51

Table 1: Sample data of Approximate Multiplier using Impact First Adder and Multiplier approximating the last four bits.

Methods

Our method involved implementing approximate hardware into the functional units of a CGRA. This included both the adder and multiplier circuit inside the ALU of each functional unit.

Furthermore, our method allows for control of the number of bits to be approximated. The programmer can specify the number of bits to be approximated. From there, the system will use approximate arithmetic operations on the least significant bits and exact approximate arithmetic operations on the most significant bits. This allows a wide range of approximation levels to be achieved using only a few approximate architectures.

Results

Two different simulation programs were explored for the simulation of our CGRA, CGRA Compilation Framework (CCF) and CGRA-ME. During this project, we chose to focus primarily on using CCF as our simulator, as it allows high level C code to be compiled and simulated on a CGRA.

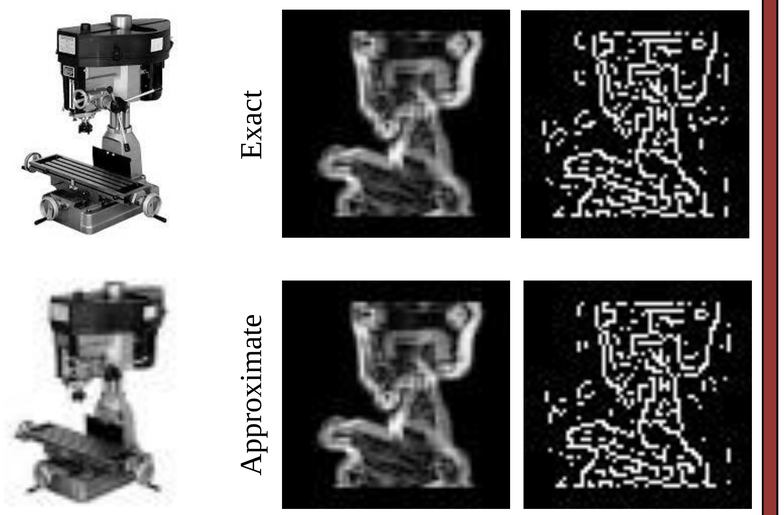

Figure 2: (Top Left) Original image, (Bottom Left) Scaled down image (Top Right) Blur and Peaks calculated using exact arithmetic (Bottom Right) Blur and Peaks calculated using approximate arithmetic

Results (continued)

Our testing used the isqrt function in MiBench as our benchmark. Our code was compiled using CGRACC, which was provided by CCF, and was then run on ARM architecture simulated by gem5. Various sizes of CGRAs were tested, including 4x4, 6x6, 8x8, and 10x10, with a variety of adders and multipliers with varying degrees of approximation.

Later testing moved to running an edge detection algorithm on an image, as this algorithm is more similar to a realistic task. Due to limitations with CCF, the edge detection algorithm could not be simulated on a CGRA. Instead the Edge Detection was simulated on the CPU.

Conclusions

By using approximate hardware instead of exact hardware, it is possible to reduce the number of logic gates required to implement addition and multiplication circuits. Although we have not done system level simulation, reducing the number of logic gates will reduce the power usage while increasing performance. Further research should be done to quantify the power reduction and performance gain.

Acknowledgements

I'd like to thank my research professor, Iraklis Anagnostopoulos and Ioannis Galanis for their assistance with this research project.

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] Zhang. Q. et al. "ApproxIt: An Approximate Computing Framework for Iterative Methods," *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, San Francisco, CA, 2014, pp. 1-6.