MULTI-LAYER APPROXIMATE COMPUTING

by

Jasleen Brar

Supervisor: Dr. Paulo Garcia

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree
Masters of Applied Science
in
Electrical and Computer Engineering

Ottawa, Ontario

© 2021, Jasleen Brar

ACKNOWLEDGEMENTS

First and foremost, I am grateful to Almighty God for leading me down the path of knowledge.

I would like to show my greatest appreciation to Dr. Paulo Garcia, my thesis supervisor. I can't say thank you enough for his tremendous support and help. His continuance support, patience, and guidance were vital for the completion of this project. Every time I attended one of his meetings, I felt inspired and driven. Without his encouragement and advice, this thesis would not have been possible.

Special thanks to students of our research group for their valuable comments and suggestions.

I am also thankful to authors of all publications whose works I have consulted for the preparation of this thesis.

I would like to express my heartfelt gratitude to my parents, Dr. Iqbal Singh and Mrs. Jasveer Kaur, for their wise counsel and sympathetic ear. They have always been source of inspiration and strength for me. Finally, I would like to thank my younger sister, Ishmeet for providing me with happy distractions to rest my mind outside of my research and my friends, Amandeep Virk and Lakshay Attri for the stimulating conversations.

ABSTRACT

Approximate computing can efficiently trade accuracy for energy savings, thus optimizing the energy efficiency of applications that do not necessarily require exact solutions. This tradeoff between accuracy and energy efficiency has been studied in recent years across several implementation technologies, but mostly focused on a single source of approximations: i.e., approximations on a specific system layer (e.g., compiler, hardware). How approximations across multiple layers of the processing stack interact to effect energy efficiency, and how these interactions affect accuracy, is still an open problem. This thesis investigates how approximations interact at different layers and affect accuracy and energy. Specifically, we present a hierarchical model that attempts to analytically define levels of approximations, such that we can model the effects of approximations at different layers, as a function of a priori knowledge of the effects of approximations at each layer. We evaluate the validity of our model for a representative image processing application across standard test images from the University of Southern California database. Approximations are evaluated across two layers (software algorithmic implementation, and arithmetic hardware, respectively), for an implementation running on a RISC-V simulator. The results for the evaluated test cases indicate that while multi-layer model of type A (linear model) provides pessimistic estimates of accuracy effects (i.e., effective accuracy is substantially higher than the predicted one), the model of type B (logarithmic function for accuracy and multiplicative function for energy) anticipates both energy and accuracy with >98 % accuracy. We discuss ongoing work towards improving this model and enhancing its evaluation across more comprehensive empirical results.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

MRI Magnetic Resonance Imaging

AC Approximate Computing

RISC Reduced Instruction Set Computing

RISC-V RISC five

FBB Forward Body Bias

RBB Reverse Body Bias

NMOS N-channel metal-oxide semiconductor

PMOS P-channel metal-oxide semiconductor

CMOS Complementary metal–oxide semiconductor

ISA Instruction Set Architecture

ADC Analog to Digital Converter

MCU Microcontroller Unit

CPU Central Processing Unit

OS Operating System

DMM Digital Multimeter

DAQ Data acquisition systems

SVS Static Voltage Supply

DVFS Dynamic Voltage and Frequency Scaling

DPM Dynamic Power Management

CNN Convolutional Neural Network

AMD Advanced Micro Devices

MSB Most Significant Bit

ARM Advanced RISC Machines

PK Proxy Kernel

RAM Random Access Memory

GNU GNU's Not Unix

GDB GNU Debugger

GCC GNU Compiler Collection

SSIM Structural Similarity

PSnR Peak Signal-to-Noise Ratio

1

INTRODUCTION

1.1 OVERVIEW

Over the past century, enormous technology advancements have aided progress in various fields. Our lifestyles have altered as a result of new devices and technologies like smart phones, WiFi[2], wearable devices[60], MRIs[77], image processing[16] and so on. The modern devices and technologies are equipped with powerful features but, aside from providing a convenient living, they come at an expense. Their computing speed is undeniably high, but so is their energy consumption. More energy usage equals more greenhouse gas emissions and more costs for us as individuals and as a society. The US Energy Information Administration (EIA) [83] forecasts a roughly 50% increase in global energy consumption between 2018 and 2050 in its newly issued International Energy Outlook 2019 (IEO2019) Reference case. With the current rate of expansion, there is a pressing need to improve power efficiency and utilization. This has been a critical topic for some time, and researchers have invented a range of power optimization techniques to address the issue. Different technologies have used techniques such as Power[66],[85],[42] and Clock gating[47],[45] at the hardware level to Parallezing or pipeling[3] at the software level to optimize power. Dynamic Power management[68], [87] and Dynamic Voltage and Frequency Techniques[41],[12] are mainly used in modern systems. The approach of dynamic power management, which refers to the selective cutoff or slowdown of system components that are idle or underutilised, has proven to be very effective in decreasing power consumption[65]. The basic concept is to place underutilised or idle resources into low- or no-power phases of operation. Clock gating is a known DPM technique in which it is possible to resume normal system activity in a few clock period.

Some of these power optimizations techniques have reduced power consumption significantly, but they come with a number of drawbacks[9]. For instance, additional circuity requirements to banish timing errors, parallelism necessitating expensive synchronization. Some techniques also require complex designs, additional feature requirements per application, costly testing methods, also not to mention that they still have the probability of becoming unreliable. When using techniques that use higher

supply voltage or lower clock frequency than required under normal circumstances, the engineers need to add extra guard bands to ensure the correct working of the device[9]. The expense of ensuring that the system behaves correctly and produces accurate results is unquestionably higher in terms of power, effort and timing.

Approximate computing, a technique has been proposed for producing efficient, low-power designs with minimal effort. The main idea of approximate computing is to trade computation time and energy for the output accuracy. By relaxing the need for fully precise or completely deterministic operations, approximate computing techniques allow substantially improved energy efficiency[33].

AC is generally implemented in the areas where certain errors in the output are acceptable, like in multimedia, search-engines, data-analytics, image processing and data mining applications. These applications are mostly guided by human perception and are error-tolerant. Approximate Computing makes it possible to create both energy-efficient and simple designs. Approximate computing approaches have been applied throughout the computational stack. Its impact has been previously evaluated at software level, circuit-level, and compiler-level, among other places.

Approximate computing techniques used at the system's software and hardware levels can have varying effects on power consumption. Authors from [75] implemented loop perforation approximate computing approach and claimed that with less than 10 percent loss in accuracy, a two to seven fold improvement in performance for multimedia, image-processing and other applications is achieved. The use of a modified inaccurate 2x2 building block in a multiplier architecture[90] with customizable error characteristics resulted in an average power savings of 31.78% to 45.4% over similar accurate multiplier designs, with an average error of 1.39%-3.32%. As per [59], combining computation and memory units when reducing the bit-width results in lowering power from 7% to 29% for Sobel filter application and from 13% to 24% for an application related to robotic arm. The trade-off between accuracy and power efficiency has been studied in recent years across several implementation technologies, but mostly focused on a single source of approximations: i.e., approximations on a specific system layer (e.g., compiler, hardware). How approximations across multiple layers of the processing stack interact to effect power efficiency, and how these interactions affect accuracy, is still an open problem.

Moreover, the evaluation and validation of techniques in approximation computing research is extremely complicated. This complication arises from the lack of a mechanism for estimating the influence on energy and ac-

curacy without resorting to time-consuming simulations. As a result, a modeling framework that allows researchers to quickly evaluate approximate computing techniques while also capturing the efficiency–accuracy tradeoff is a must. REACT[89] is such a tool based on a simple linear energy model that measures the effects of approximation techniques on applications. In [82], a probablistic approach is used to investigate the impact of precision scaling strategies on accuracy, and the results demonstrate that the proposed strategy can estimate the approximation error with a high degree of accuracy (98 to 99 percent) with very low computation time. To the extend of our knowledge, currently there are no models available that provide an early indication of the energy savings that can be obtained by combining approximation computing technologies at different stack levels.

In our work, we first establish that using approximation computing techniques improves performance and energy, and that combining them can benefit us even more. It is presumed that updating any layers of the system stack is not constrained. The main focus of the thesis is to provide estimation of energy savings and accuracy when multi-layer approximate computing techniques are implemented. To do so, we present a multi-layer approximate computing methodology that starts with an analytical energy-accuracy model to derive the results. As part of the modeling methodology, we present equations to determine the energy-accuracy relationship of techniques at multiple layers. A RISC-V functional simulation-based analysis is then used to evaluate and refine this theoretical model.

1.2 THESIS CONTRIBUTION

This thesis presents our work on modeling multi-layer approximations, towards a framework for strategic applications of approximations with emphasis on image processing domain. To begin, a survey of various approximate computing techniques is offered, both at a single layer and at multiple layers. This is to give an indication of how well approximate computing approaches have worked in reducing energy consumption without much impact on accuracy.

Secondly, error profiling and characterization has to rely on simulations without the help of theoretical error analysis. A good analytical model is therefore required to either guide early design or provide more information for higher-level hierarchy. We describe a model for multi-layer approximations. The model predicts how approximations at different stack levels affect energy and accuracy. The model is built on the idea that if we know

the impact on separate layers, we can simply analyze the impact when approximations are used together using the model.

We also describe the model's application to an image processing benchmark, across hardware and software. To test our theoretical model, we performed experiments on an image processing application, Sobel filter, using a RISC-V simulator. We implemented approximate computing techniques at algorithmic and architectural level. The approximate computing technique at the algorithmic level is loop perforation and that at architectural level is bit-width reduction. Experiments are performed to evaluate energy and accuracy trade-offs when two approximate computing techniques are implemented simultaneously. Our results indicate the model predicts energy savings successfully within a 1% difference, but pessimistically estimates resultant accuracy. However, the model is modified based on empirical results to precisely anticipate the accuracy tradeoff. The new model performs better, with accuracy predictions within 2% of the simulation results.

At last, we discuss ongoing work towards improving this model and enhancing its evaluation across more comprehensive empirical results. While the empirical results achieved are encouraging in terms of the applicability of multi-layer approximations to a wider range of applications, additional work is needed to improve modeling outcomes. The limitations of the current model are first outlined, and the improvement in model efficiency that can be obtained by adding specific components is then discussed at length.

1.3 THESIS ORGANIZATION

The thesis is organized in six main chapters as follows:

Chapter 2 discusses how energy and power is consumed and measured, as well as what approximation computing entails. This section also describe the RISC-V architecture targeted for our work. Finally, a summary is offered that leads to the selection of the RISC-V simulator, Spike.

Chapter 3 discuss the related work to the thesis contributions. The first section presents power optimization techniques and their limitations, followed by approximate computing techniques at software and hardware layer and at multiple layers. We also present several power-accuracy models used in various studies for evaluation.

Chapter 4 describes our experimental testbed, the architecture of the RISC-V processor and Spike simulator extended with approximate instructions. Then we show how we use a multi-layer approximate computing model to estimate the energy consumption and accuracy.

Chapter 5 presents our experimental results obtained implementing approximate computing at software and hardware level on a Sobel filter application. We compared these results to those produced with our multi-layer model and discussed them. Our revised model based on empirical results is also proposed in this section.

Chapter 6 concludes the thesis. At first the main contributions are summarized, then suggestions for future work are provided.

2

BACKGROUND

2.1 ENERGY AND POWER CONSUMPTION

2.1.1 *Overview*

In today's electronic devices, power and energy have become crucial factors. In the literature, they are often used as synonyms, though there are underlying distinctions between them. While energy is the ability to do work, power is the measurement of how much energy has been consumed over time. The necessity to regulate heat dissipation of increasingly stronger processor, high supply costs and energy consumption costs make it difficult to disregard energy and power consumption. Understanding and optimizing the device's power consumption is one of the most significant factors in the product life cycle of any electronics project. In order to optimize energy or power usage in any equipment, first we need to collect information on the power consumption of the device. Since most digital circuits are currently implemented using CMOS technology, it is reasonable to describe the essential equations governing power and energy consumption for this technology.

There are several factors contributing to the power consumption of a CMOS circuit; they include dynamic power consumption, short-circuit power consumption, and power loss due to transistor leakage currents. So, total device power is calculated as:

$$P_{total} = P_{dynamic} + P_{static} + P_{sc}$$

$$P_{dynamic} = \alpha * C * V_{dd}^2 * f$$

$$P_{sc} = \alpha(\beta/2)(V - 2Vth)$$

$$Pleakage = (I_{diode} + I_{subthreshold}) * V_{dd}$$

where α=Switching Activity, C=Total Load Capacitance, V_dd=Supply Voltage, f=Frequency, β=Gain Factor,T_rf=Rise/Fall Time, V_th=Threshold Voltage. .

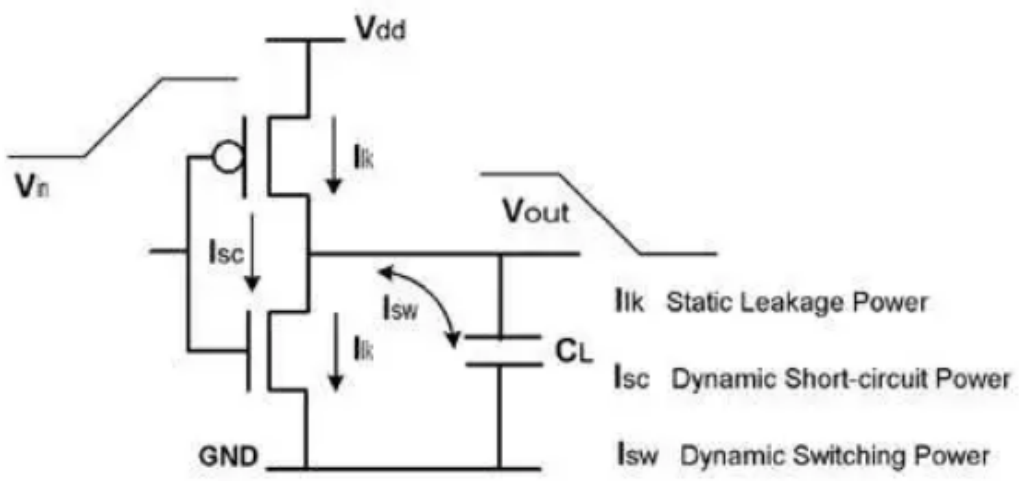


Figure 1: CMOS Power Dissipation, Source:[29]

Static power consumption or leakage consumption is the power that a device consumes independent of any activity or task the core is running because even in a steady state, there is a low leakage current channel from the device's supply to ground that results in static power consumption. Supply voltage, temperature, and process technology are factors that influence leakage consumption.

Dynamic power, on the other hand, originates from the activity of logic gates inside a CPU[88]. At the low level, it can be implied that dynamic power is the amount of energy consumed by switching transistors that charge and discharge capacitances. More cores, more arithmetic units, more memory, higher clock rates, or anything else that could potentially increase the number of transistors switching, or the speed at which they transition, increases dynamic power[79]. The dynamic power consumed, is approximately proportional to the CPU frequency, and to the square of the voltage.

The contribution, the short-circuit power, is related to the short-circuit currents flowing through the MOS transistors in the gate at each switching. It is strongly dependent on the parameters present in equations(switching activity, clock frequency, and supply voltage), but it also depends on the design of the circuit.

Since software directs much of the activity of the device, aside from the hardware, it can have a significant impact on computational systems' power dissipation. But the software's power consumption is only related to dynamic power consumption rather than static power consumption. Energy consumption reduction of an embedded processor can be attempted in many different ways viz. hardware design approaches and software approaches, discussed in power optimization techniques section.

In general, the frequency, voltage, kind of application, and process technology utilised by the device are the four key parameters that influence power consumption in every device[79]. The relationship between power and frequency, as well as the effect of voltage on power, is evident from the equations. The higher the supply voltage, the higher is the power consumed by the device. As the supply voltage and frequency of switching operations in the circuit rise, so does the power dissipation.

Because power consumption changes based on the application, two devices with different power profiles, such as an ECG and a cellphone, will have quite different power profiles. This is why application power profiling is so frequent before attempting to optimize power consumption. As far as process technology is concerned, it can reduce leakage power by more than an order of magnitude and also has a large impact on dynamic power[18]. The new cutting-edge technology employed small transistors which consume less power and so are clearly better.

2.1.2 *Measuring Energy/Power*

There are three main methods in order to estimate the energy consumption of any device:

- **Measurement based energy estimation:** Instruments are a simple way to measure the system's energy usage on the hardware circuit. As shown in Fig.2, the basic framework for power measurement includes a high-precision shunt resistor connected between the power supply module and the device. Instrument probes, like DMM, DAQ are attached to both sides of the resistor, sampling the voltage across it and the server retrieves the relevant data for analyzing energy usage. However, for these instruments, the expensive cost of instruments, their bulky size, and the difficulty of developing flexible measurement studies are a problem. Power measurement platforms are employed to address these concerns[31].

  Fig.3 depicts the system design of such power measurement platform. It has a shunt resistor connected between the power supply and the load. To amplify the voltage drop across the shunt resistor and provide it to the ADC, a current sense amplifier or differential amplifier is required. The signal is then sent to the MCU, which is in charge of gathering data to directly process it or sending it to the computer to determine the power utilization.

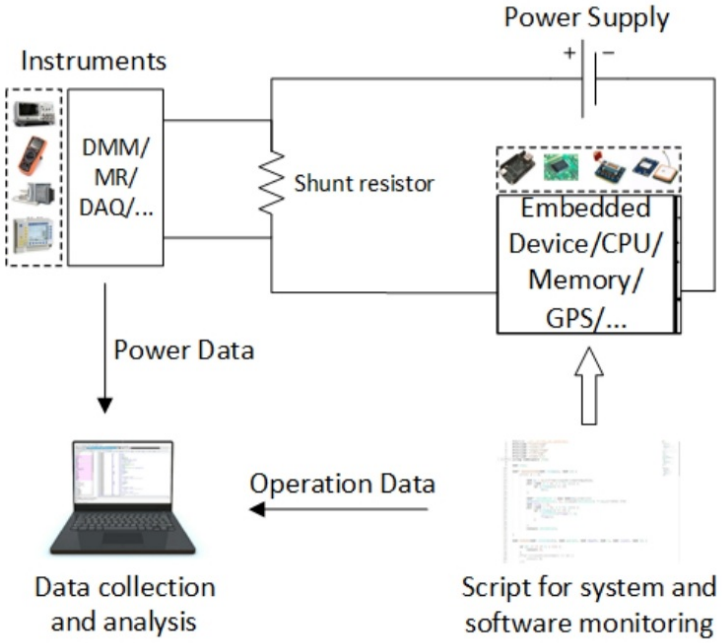


Figure 2: Basic framework of power measurement, Source:[31]

The platforms are inexpensive and compact, and capable of providing tailored analysis based on measurement goals. But such a platform has a limited measuring range due to the difficulty and cost of circuit construction, making fine-grained energy analysis nearly impossible[31].

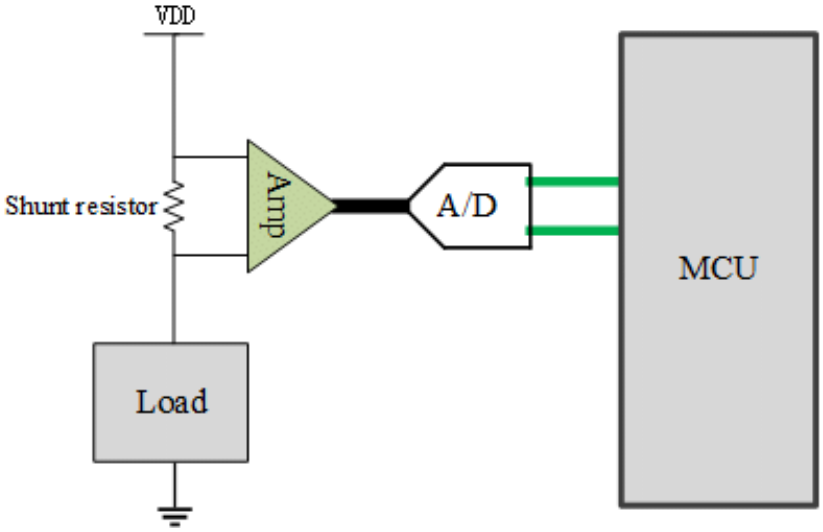


Figure 3: Design of power measurement platform, Source:[31]

- **Modeling based energy estimation:** Analytic models are a useful tool for constructing theories. They have been intensively researched

in order to define concepts and explain phenomena. It enables readers to critically examine theoretical assumptions, builds a relationship between the research and prior knowledge, and allows the researcher to progress from just reporting a phenomenon to generalizing about multiple aspects of that phenomenon. It's incredibly beneficial to have a general validated model for computing any measure. It not only saves time, but it also gives us a set of criteria to use when analyzing it. Several researchers have proposed various energy models, including those for approximate computing techniques. Some of the energy models are discussed later in Related Work section.

- **Simulation based energy estimation:** Simulation is considered to be the standard performance modeling method[76]. Simulation is used to evaluate various design possibilities, study different models, and to test new research ideas. A simulator is a tool used to conduct simulation in embedded systems. It is a collection of hardware and software systems which are used to mimic the behaviour of some entity or phenomenon. The majority of published research is based on the use of simulators to analyze the performance of new ideas. Many computer architecture simulators support various instruction set architectures (ISAs) and micro-architectures.

  The main reason for choosing simulation over testing in a real-world is because simulations are portable, easy to distribute, and the results can be easily reproduced. Simulators also provide a lot of flexibility because we don't have to worry about changing system configurations or building new hardware prototypes. Without a simulator, applications can almost certainly be designed, tested, and debugged. However, there are a number of reasons why using a simulator can make work easier and save a lot of time during development. Some of these reasons include simulators being cost effective, to analyze theoretical models, to train, to easily explain complex problems and to analyze performance of next-generation systems.

2.2 APPROXIMATE COMPUTING

In an era when present technologies appear to deplete available resources, approximate computing is the answer. While exact calculation requires a considerable number of resources, allowing bounded errors or, in other words, approximation on purpose can result in significant power and performance advantages.

Much of our compute power is spent on sensing, measuring, and guessing and since sensing is noisy, measuring is imprecise, and guessing is imperfect, we can save a lot of energy by accepting approximation[22]. Instead of employing floating-point precision for computation, a reduced precision is usually acceptable since the inputs to those applications typically contain inaccuracies that are much larger than the errors resulted from approximation. Certain applications, on the hardware side, are tolerant to faults or noise and could therefore profit from faulty hardware and the energy savings that come with it. For instance, digital signal processing uses faulty hardware that exploits approximation via quantization and decimation.

Approximate Computing is based solely on the acceptance of error-tolerant applications and on the limitations of human perception. Applications in computer vision, machine learning, image and video processing have built-in tolerance for error. There are no perfect answers in domains like computer vision and machine learning and the output quality is largely dependent on the available resources. Because humans have limited perceptual capacities when interpreting an image, sound, or video, the image processing system can be flexible in providing quality outputs.

Approximate computing simply means going beyond traditional performance restrictions by embracing approximation in different tiers of the system stack. In recent years, various approximation techniques, such as faulty hardware, memoization, numerical approximations, loop perforation, and neural network accelerators[57] have been applied at the algorithmic, architecture, and circuit levels. Several of approximate computing techniques are briefly reviewed in the following chapter.

It should, however be noted that Approximate computing is limited to approximate applications and cannot be used in areas where precision is a necessity. In AC, maintaining appropriate output quality is also a challenge. When using any approximation strategy, it's crucial to define an error margin. We are trading error for power, but the amount of error that an application can accept is something that must be considered. There have also been tools[63],[71],[70], [89] developed to assist in the implementation of AC procedures as well as in varying the error margins to determine the acceptable output quality. Different quality metrics can also be utilized to determine the quality of the produced output, such as PSNR in case of image processing.

2.3 RISC-V ARCHITECTURE

To facilitate the implementation of multiple level approximate design, integration at an architectural level is also a requirement. RISC-V can assist in bridging the gap between software and hardware in bridging the gap between software and hardware approximations and making it easier to evaluate the trade offs between energy and accuracy[25].

In simple words, an instruction set is a computer language vocabulary that specifies a computer's design in terms of simple operations like arithmetic and logic instructions. It's the interface between the hardware and the lowest-level software that contains all of the data necessary to construct a machine-level program. The main difference between different ISAs on the market is the ease with which they can be designed and the environment that surrounds them.

RISC(Reduced Computer Instruction Set Computing) and CISC(Complex Instruction Set Computing) are the two basic types of processors. RISC-V is a general purpose architecture that serves as a basis for a variety of projects from industry and academia. We chose RISC-V in particular because of its advantages. While RISC-V is not the first ISA based on the RISC principles, one of its biggest benefits is its open source nature, which enables customization that is not possible with other ISAs. Most of the authors, vendors and other consumers would agree that its open source nature makes it easier to adapt, expand and implement. The fixed user-level ISA also ensures software compatibility and longevity of the architecture[86]. Moreover, the base RISC-V ISA is not only simple to implement and maintain but also is comprehensive enough to handle modern software stack, making it appropriate for our work.

An another feature of RISC-V is modularity which allows adding new instructions as optional extensions instead of releasing new versions of the whole ISA[69]. RISC-V effectively has three base instruction sets and six extensions. Although the RISC-V base ISA and the prespecified extensions are fixed length, some extensions are not (e.g.,compressed instructions). The ability to branch RISC-V to better meet certain use cases is definetely something that sets it apart from the competition.

In addition, it is suitable for direct hardware implementation, simulation and for binary translation. To evaluate approximation computing results for the thesis work, simulation-based implementation is used.

2.4 RISC-V INSTRUCTION FORMATS AND OPCODES

Different instruction set formats are supported in RISC-V, some of these include R-type, I-type, S-type, U-type format. All are 32 bits long and must be aligned in memory on a four-byte boundary[86]. The R-type format

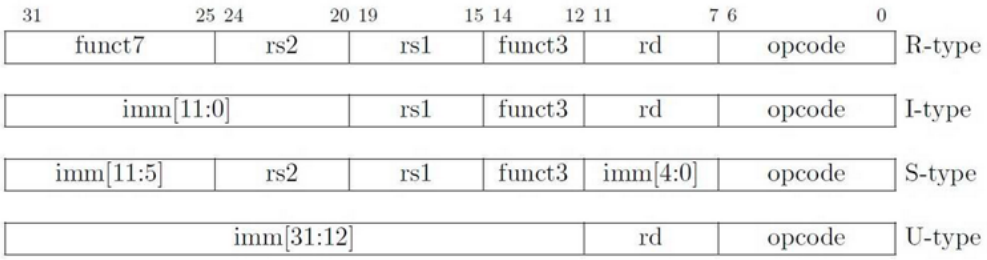


Figure 4: RISC-V instruction formats, Source:[86]

uses two input registers and one output register like add, xor, mul. For the R-type format, the bits, 31-25(funct7) along with bits, 14-12 decide what operation to be executed. rs1, rs2 and rd represent the two source registers and destination register respectively. The I-type format has only one source register and has a imm[11:0] field in the bits[31 : 19] for the immediate values to compute with the value of the register rs1. From 19 bits and below, the I-type and the R-type have the same formats. The S-type format has two source registers and an immediate destination, e.g. conditional branch instructions. The only way it is different from R-type format is that the fields imm[11:5] and imm[4:0] correspond for the immediate values in S-type instructions. The U-type format has 20-bit immediate in upper 20 bits of 32-bit instruction word and one destination register, rd. It is used for two instructions, LUI(Load Upper Immediate) and AUIPC(Add Upper Immediate to PC). In this work, the main focus is on R-type instruction format. Fig.5 shows all R-type instructions in 32-bit RISC-V architecture.

0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

Different encoding in funct7 + funct3 selects different operations

Figure 5: R-type instructions, Source: [86]

Each of the instructions is associated with an opcode. RISCV standard extensions have been reserved with some opcodes. In case one need to add any custom instructions, none of the standard instructions opcodes can be used. Any of non-reserved opcodes can be utilised for new instructions, the recommended opcodes are custom-0 and custom-1.

2.5 RISCV SIMULATORS

To evaluate our approximate computing model and empirically measure energy consumption, a RISC-V simulator is required. A simulator employs a computer program to create a model of a real system. When the program is executed, the mathematical dynamics that arise constitute an analogue of the real system's behaviour, with the outcomes shown as data[81]. RISC-V simulators are used to simulate the RISCV architecture without needing actual hardware. There are a wide range of simulators available, from functional to cycle-accurate.

Functional simulators are primarily meant to model big and complicated systems as quickly as feasible to enable efficient functional validation. They show the designers the highest level of simulation accessible. In a functional-level simulation, an abstract unit would accept input and produce output in the same way that its corresponding hardware component would[20]. The goal of these simulators is to provide the designer a high-level perspective of the design and allow them to experiment with various options. While functional-level modeling tools can be very fast, they do not provide detailed timing information. On the other extreme, cycle-exact models can provide very detailed and accurate timing information. They may simulate a processor's instruction set, pipeline, and local cache, as well as deliver signals at the CPU's pins at each clock transition with exact clock cycle counts. It can also be used to model interaction with physical components in addition to software simulation[27]. However, they are more than an order of magnitude slower than an instruction set simulator and because of their complexity, they are difficult to modify.

Before choosing Spike for the testing purpose, we ran a pre-evaluation of various RISCV simulators, namely riscvOVPsim[95], gem5[93], QEMU[67], Spike[78] and FireSim[26]. riscvOVPsim[95] is a free closed source simulator binary from Imperas that requires no compilation or other dependencies to run. The model was created for personal, academic, and commercial use, and it is open source under the Apache 2.0 licence. gem5[93] is a community-driven project with a open governance structure and can be modified. It was created for academic computer architecture research, but it

has since expanded to include computer system design in academia, industry research, and teaching. QEMU[67] and Spike[78] are both open-source functional RISCV simulators designed for ease of use whereas FireSim[26] is an open-source FPGA accelerated cycle-accurate RISCV simulator actively developed in the Berkeley Architecture Research Group.

The pre-evaluation of the simulators is based on several elements such as support for tracing instructions, full-system OS support, and so on. The table 1 summarises the findings. Gem5 is certainly the only simulator that checks all of the boxes, and it should be the obvious pick. However, because gem5 is a cycle-accurate simulator and in comparison to a functional simulators, it would be difficult and time-consuming to modify. Our testing framework's major demand is customization and faster simulation results. Therefore to assess our multi-layer model and calculate performance, we used Spike, a functional model of RISC-V harts which can be easily extended to add new instructions. Spike has a number of characteristics that make it ideal for this thesis project. It's a modular platform, similar to RISC-V, with numerous micro-architectures that may be customized to meet the needs of the user. The riscv-isa can be correctly executed by the simulator. It provides both full system and proxy emulation. In full system mode, it emulates all of the hardware from the CPU to the I/O devices. In full system mode, as a user, we have to provide a compiled Linux kernel and a disk image. Then, to run applications in simulator, the operating system has to be booted and then we can interact as if it is a running computer. The running application code can be compiled with standard tools and using standard C libraries without modifications. There is no formal documentation for the simulator, and understanding and then modifying/adding new capabilities takes time, although the simulator's community forum is extremely helpful. Apart from offering multiple ISAs models and multi-core support, its interactive debug mode is a great addition.

Table 1: Pre-evaluation of RISCV Simulators

	Spike	Qemu	gem5	OVPsim	FireSim
Type of Simulator	functional	functional	cycle-accurate	functional	cycle-accurate
License	Open	Open	Open	Open and Private	Open
Trace instructions	✓	✗	✓	✓	✓
Flexibility	✓	✓	✓	✓	Requires RTL/abstract models
Checkpointing	✓	✓	✓	✓	✓
Multicore Support	✓	✓	✓	✓	✓
Modular	✓	✓	✓	✓	✓
Power/Energy Simulation	✗	✗	✓	✗	✗
Full-system OS support	✓	✓	✓	✓	✓

3

RELATED WORK

This chapter begins by outlining general power optimization approaches before summarising a representative set of existing work on approximation computing at both single (Software and Hardware) and multi-level levels.

3.1 POWER OPTIMIZATION TECHNIQUES AND THEIR LIMITATIONS

Several researchers have focused their work on reducing power consumption on the hardware and software side of computational systems. The most direct way to reduce power is supply shutdown, which is an optimization technique that eliminates all causes of power loss, including leakage. As the name specifies, it involves shutting off a component's power supply. However, the component must be reinitialized in a supply shutdown strategy, and the wake-up recovery period is really long, which is a significant disadvantage. It also has a number of other major flaws, including current spikes on power and ground lines and data loss in memory cells.

One of the most common and successful techniques used is clock gating, which prevents power consumption by shuting down a functional unit. The power utilization by spontaneous switching is prevented as clock signal stops whenever this functional unit becomes idle. This power management technique is very successful as it only takes one or two clock periods to resume the normal system activity[7]. Different gating methods like NOR Gate, Latch Gate or AND Gate can be used to implement clock gating but latch enabled gating (shown in Fig.6) is better as compared to the circuits without latches as it does not expose the circuit to glitches[47]. There are a few challenges associated with clock gating technique. To begin with, it is quite a challenge to decide which unit to shut down. This technique also results in possible timing violations and testability degradation. Moreover, it should be noted that power dissipation is not totally eliminated by clock gating. The clock-generation circuitry is still operational and dissipates power. To solve timing limits and clock skew concerns, [45] developed an automated layout design technique for the gated-clock.

While clock gating focuses on the circuit's dynamic power by lowering the switching frequency, power gating focuses on the circuit's static/leakage power by lowering the current flow through the circuit. The purpose of

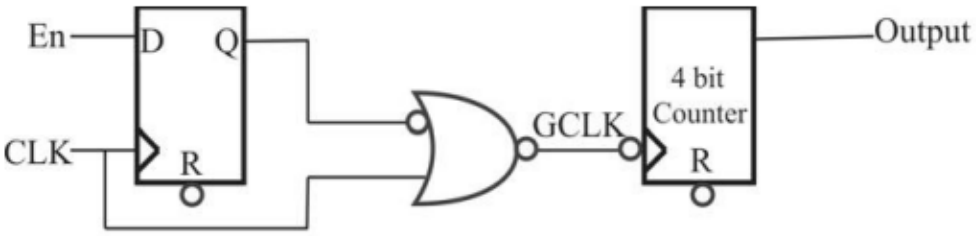


Figure 6: Clock gating using latch-based NOR gate, Source: [47]

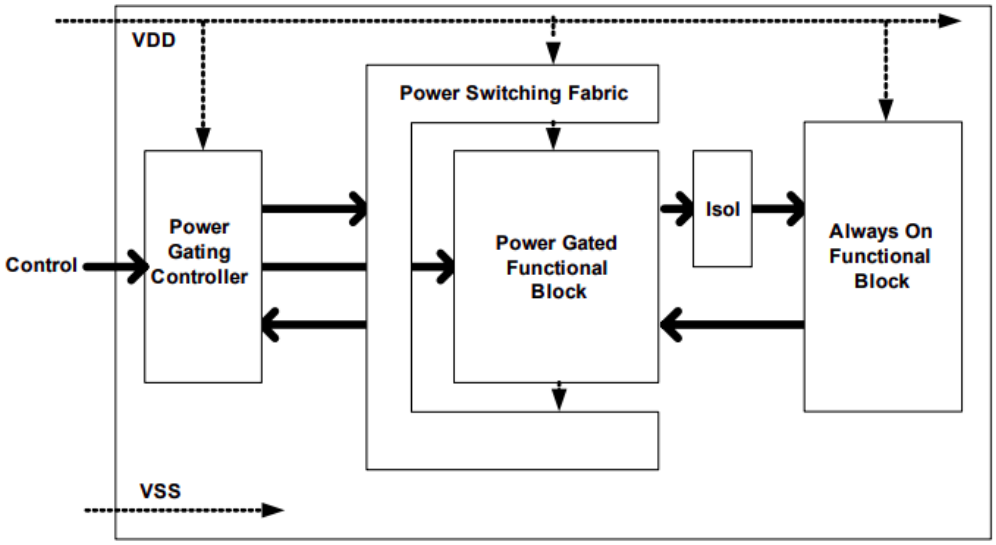


Figure 7: Block Diagram of an SoC with power gating, Source: [66]

power gating is to switch between a low power mode and an active mode at the appropriate time and in the proper manner to maximize power savings while minimizing performance effect[66] using a power-switching network. For instance, in the block diagram, shown in Fig.7, VDD(Drain Voltage) is switched while VSS(Source Voltage) is provided directly to the entire chip. [85] examines several power gating approaches based on a variety of parameters and concludes that Dual-Switch Power Gating offers the most benefits. However, in order to maintain state and information during low power mode, all power gating systems require additional functionality. Other negative impacts of power gating include a combination of noise, performance penalty, area and power overhead, etc[42]. When power and clock techniques only reduce dynamic or static power, there are some other power optimization approaches that can lower both dynamic and static power.

To ensure correct and dependable functioning under all operating conditions, including temperature and frequency, the core operating voltage standards stated in the datasheet of devices and processors must be strictly followed[94]. By optimizing using supply voltage scaling, we can, however, control leakage, reduce delay and save a large amount of power. Static volt-

age scaling is one of the voltage scaling optimization techniques in which distinct blocks or subsystems receive fixed supply voltages. But this technique can only be applied on parts that are built particularly for SVS functioning. Additionally, optimal voltage scaling algorithms are costly.

Multiple voltage scaling technique is an extension of SVS, in which a block or subsystem is switched between two or more voltage levels, with only a few fixed, discrete levels supported for various operating modes [48]. This technique not only complicates optimization, but it might also impede power savings. Between the low supply voltage and high supply voltage gates, a voltage level converter must be installed. Level converters require additional power and delay, which must be encoded in the linear program. The main limitation of power-driven voltage scaling is that it becomes less effective as technology scales down.

Body Bias is a design method that can be utilized to significantly reduce delay and leakage. Body biasing scales the threshold voltage of a device without varying the power supply. The threshold voltage can be increased or decreased depending upon the polarity of the voltage difference between the source and the body terminal. Forward body bias scales down the threshold voltage and reverse body bias increases the threshold voltage. Thus by wisely applying the FBB and RBB we can reduce the delay and leakage of the design. Fig.8 and Fig.9 shows RBB and FBB applied on NMOS and PMOS transistors respectively. [35] proposed a noval design of pre-computation-based adaptive leakage control employing body bias which lowered the leakage power of the circuit is by up to 20%. [58] also showed how adaptive body bias can be used to reduce delay and leakage. The main disadvantage of this technique is that it puts more strain on a device, which might cause it to degrade over time, shortening its intended longevity and/or performance. It can also only be used on components with additional circuitry that are intended to be in a body bias state.

Dynamic voltage and frequency scaling (DVFS) [41] is a technique for lowering dynamic power consumption by optimising resource workloads tasks by varying the voltage and frequency of a CPU dynamically. This technique exploits the fact that CPUs have discrete frequency and voltage settings[12]. Fig.10 shows a block diagram of DVFS design. The CPU subsystem is powered by a configurable power supply, whereas the rest of the chip is powered by a set supply voltage. A Phaselocked Loop (PLL) also provides a high-speed clock to the SysClock Generator, which generates the CPU CLOCK and SOC CLOCK using dividers. DVFS needs the circuit to be partitioned with specified power limits and desired clock frequencies for each area of the system. Software first identifies the minimal CPU

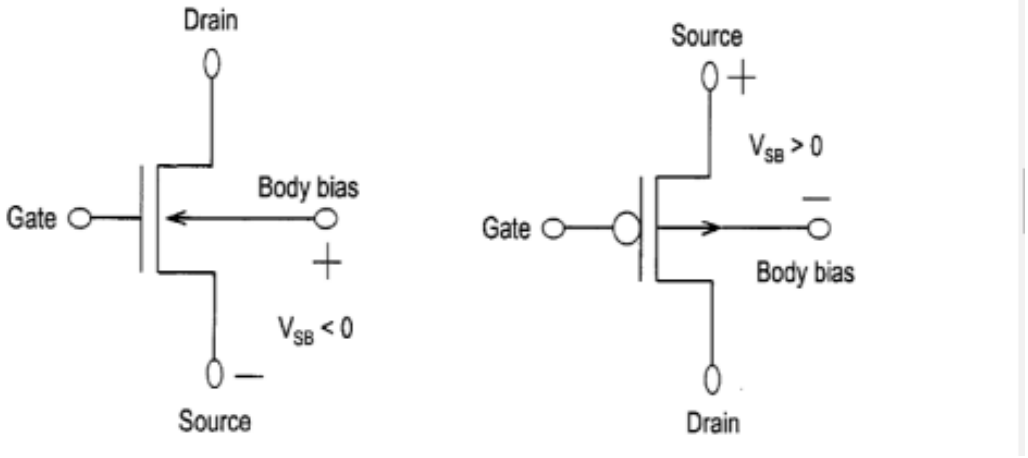


Figure 8: Forward Body Bias, Source: [47]

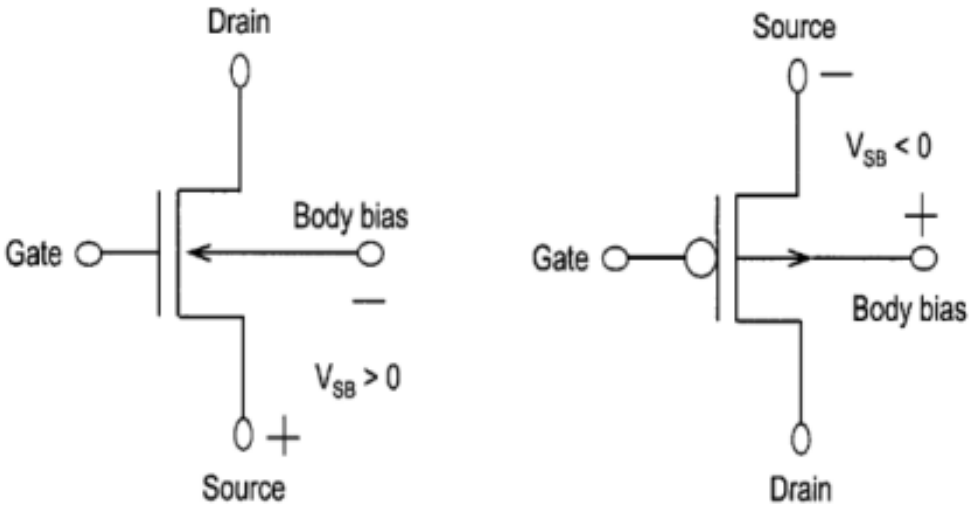


Figure 9: Reverse Body Bias, Source: [66]

clock speed, and then the lowest supply voltage that will support that clock speed while meeting workload requirements[66]. Since DVFS optimizes both the frequency and the voltage, it is one of the only techniques that is highly effective on both dynamic and static power. However, since this technique trades performance, it is important to have an accurate evaluation of the system's efficiency at different frequency settings before implementation[12]. Besides, [49] investigated the potential of DVFS on three recent versions of AMD Opteron processors, and their findings show that DVFS' effectiveness has been significantly diminished.

An another efficient method is dynamic power management (DPM), which entails changing device states when they are not operating at full speed or capacity. Since the system is not always utilizing all its resources all the times, it can be shutdown when not in use. Now the question is, when

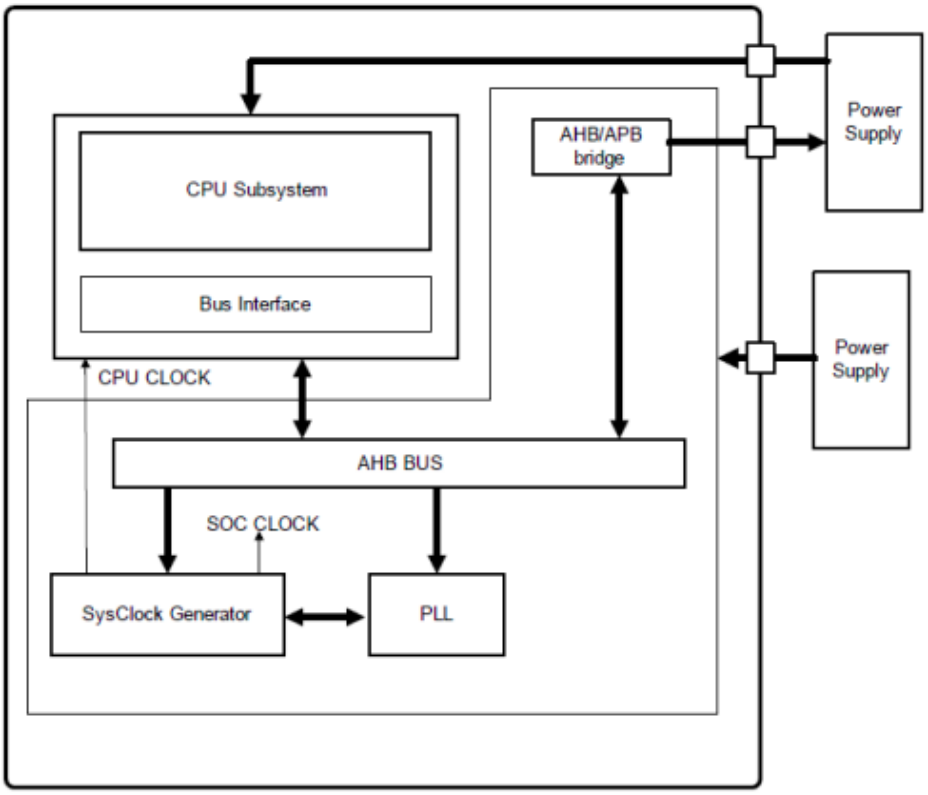


Figure 10: Block Diagram of DVFS design, Source: [66]

should the system be shut down? Heuristic, Stochastic and Predictive techniques are used to model the system and the workload to determine the time of shutdown[87]. Adaptive and non-adaptive time-out approaches are used in heuristic processes. In the case of adaptive time-out, the time period after which the system falls idle is fixed, whereas in the case of non-adaptive time-out, it is not. The problem with this strategy is that if the time estimate isn't precise, it will waste rather than conserve power, and there is still some power dissipated continually during the waiting period. Predictive policies predicts the length of the future idle period, allowing us to decide whether or not to sleep based on whether the estimate is larger or less than Tbe(break time). Similar to heuristic procedures, if the prediction is inaccurate, power is lost rather than saved in this situation as well. Instead of attempting to reduce uncertainty through prediction, power management optimization in the stochastic approach is investigated using controlled Markov processes. Markov processes have a limited memory and solely rely on the prior state for information[40]. However, while the power values are expected, outcomes are not a guarantee and may vary according to the use-case.

Dynamic Power Management approaches are still being tested, and it's worth mentioning that if a system's design is complicated, DPM can be a time-consuming process requiring design iterations and debugging[68].

Power reduction is generally associated with reducing power at hardware level but the functionality of memory and the software program that consume power can also be optimized. Memory optimization is a popular method that aims to maximise the locality of frequently used data and code by putting as much as feasible in cache.[62]. [39] demonstrated an energy-saving memory management technique that separates a large memory space into smaller blocks that may be turned on and off separately. When there are a small number of memory banks, power consumption for memory access is minimized. A high number of small banks, on the other hand, might increase wire overhead complexity and make the area inefficient, demanding more power for communication.

At the compiler level, to improve the datapath performance, instruction scheduling or reordering instructions is frequently employed. Several attempts have been made to optimise the instruction schedulers' energy consumption. [43] showed that software optimizations can be applied at different stages of the compiler, either at the high-level or the low-level. In [46], challenges connected to compiler power management are explored, as well as an analysis of three compiler approaches, namely hibernation, remote task mapping and DVFS is conducted. Lack of much accurate knowledge about control flow and program values at time of compiler optimizations make the tradeoff decisions different and challenging. Moreover, compiler research has been impeded by the lack of a dependable and effective assessment infrastructure for power and energy optimization[46].

Instruction Packing technique[74], which enable fitting maximum code into a minimum set of space can also be used to reduce delay and power consumption. Software Pipelining is a algorithmic level power-saving approach in which a number of independent instructions that would normally be handled one at a time by the processor are executed in a single cycle. The basic idea is to break the problem down into stages, with the output of one feeding into the input of the next. It starts working on the input as soon as it is accessible, even if the next stage is still running. It is often used in combination with loop unrolling and this combination together is a better optimization technique. Loop unrolling is a common loop transformation technique in which the loop body is repeated K times and the loop iteration space is minimised by unrolling a loop by a factor of K[13]. Fig.11 shows a fully unrolled loop whereas Fig.12 shows a loop unrolled by a factor of 2.

```
for(i=0; i<4;i++) {
  c[i] = a[i] + b[i];
}
```

```
c[0] = a[0] + b[0];
c[1] = a[1] + b[1];
c[2] = a[2] + b[2];
c[3] = a[3] + b[3];
```

Figure 11: loop unrolling by a factor of K, Source: [13]

```
for(i=0; i<4;i+=2) {
   c[i] = a[i] + b[i];
   c[i+1] = a[i+1] + b[i+1];
}
```

Figure 12: loop unrolling by a factor of 2, Source: [13]

The importance of software pipelining is highlighted by the wide range of architectures that benefit from it. There are a variety of algorithms for achieving software pipelining, and a comparison is presented in [3]. The matters with using software pipeline with loop unrolling, however, is generally complicated when the number of iterations is unknown. A code to test if the number of iterations is greater or equal than the unrolling factor has to be included.

Finally, irrespective of the benefits provided by numerous accurate power optimization strategies, power consumption concerns still persist, leading us to use AC which trades accuracy for power.

3.2 SOFTWARE AC TECHNIQUES

Software Approximate Computing attempts to modify algorithms, compilers or tool chains to minimize the computational work.

Loop perforation is one of the known software approximation technique which skips some iterations of the loop. Essentially, the method involves running only a portion of iterations to reduce computing work. A decent amount of work related to loop perforation has been performed and its utility in a variety of applications has been demonstrated. Paper [75] implemented loop perforation approximate computing on applications (Blackscholes, Bodytrack, etc) from PARSEC benchmark suite using a space exploration algorithm. This algorithm takes an application as input and creates a set S of loops to perforate at specified perforation rates based on specific criteria such as accuracy metric and perforation rate. With less than 10 per-

cent loss in accuracy, the results show a two to seven fold improvement in performance for multimedia, image-processing and other applications.

An another approximation technique, selective dynamic loop perforation is introduced by [53] to improve classic loop perforation method while [55] introduced an optimization technique called image perforation that transforms loops over the image such that specific samples are effectively skipped. The technique described in paper [53] is based on offering higher performance by missing particular instructions at a subset of loops. In other words, it allows to bypass only those instructions in the loop that aren't crucial to output accuracy, hence improving the overall performance. Dynamic loop perforation, however, requires a thorough examination of all of its instruction samples prior to implementation. It poses a significant challenge in capturing the features of these instructions, necessitating customized compiler optimization.

[55] introduced image perforation, a technique for investigating the performance-accuracy tradeoffs in image processing. When performing image processing for any digital image, image pipelines are the multiple stages that exist between the image source and the image renderer. Image perforation method is implemented to skip certain samples of an image by converting the loop at the pipeline levels. Results for different image pipelines and inputs indicate speedups of 2×–10× with acceptable loss. The authors of paper [55] also compared the results of image perforation to those of paper [75], finding that image perforation outperformed loop perforation.

Memoization is an another general promising technique that collect previous operations' results to improve power and performance efficiency. In the approximate computing paradigm, memoization is a method that not only associates previously computed outputs for the identical input entries but also for the ones that are nearly equivalent. This technique has been expanded by [4] as tolerant memoization to evaluate and record energy boost for multimedia applications. It uses a look-up table that stores the results of floating-point operations, together with the values of the source operands. Memoization is implemented by removing N significant bits from the mantissa of each operand prior to accessing the table. Results obtained from four different key-domain multimedia programs indicate energy improvements of 12 percent with a look-up table of merely 6 Kilobytes. However, one limitation of this method is that the quality of the data is really poor when there are significant losses.

In order to minimize the number of calculations in image and video processing applications, [23] used memoization method by applying local operations to successive frames. The approach is based on the fact that, in

most cases, the numerical values of all neighbouring frame pairs are either identical or almost identical. When executing local operations, the similarity between neighbouring frames in a video sequence, by selecting a larger number of d-last MSBs can therefore be leveraged to improve performance. The homogeneity of 2-D-histogram between adjacent frames is the used as a measure of efficiency in paper [23]. The experiments were carried out on median and edge detection algorithms, with speedups ranging from 2.2x and 2.1x without any errors to 6.48x and 5.41x for median and edge detection operations, respectively. An average SSIM of 0.96 for median and 0.71 for edge detection algorithm is detected. It should be noted that loosening the equality matching constraint between consecutive frames improved computational performance but also lowered the SSIM index, especially in case of edge detection algorithm.

Likewise, [44] proposed window memoization which blends memoization techniques in software and hardware with data redundancy, by recognizing similar areas of pixels in an image and skipping the unnecessary computations. Techniques such as Precision downgrade [17] and selectively skipping tasks [15] has also been proposed, including floating to fixed point conversion [36] and programming and compiler systems for approximate computing such as EnerJ[70], Flexjava [63] and ACCEPT [71]. EnerJ adds type qualifiers to the Java language to distinguish between approximate and exact data types. Data annotated with the qualifier "approximate" can be saved and computations involving it can be performed approximately. EnerJ's assumed hardware provides approximation in functional units via supply voltage reduction, Main memory (DRAM) via refresh rate reduction, and Floating point functional units via lower mantissas, resulting in overall energy savings of 10 to 50 percent. Similarly, Flexjava [63] is a language model extended to Java that leverages automated program analysis tools to make approximation programming more successful. It uses same approximation strategies as EnerJ. The paper compared FlexJava to EnerJ, finding that FlexJava saves the same amount of energy while lowering the number of annotations from 2x to 17x. ACCEPT[71], on the other hand, is a compiler framework that provides approximation using C/C++ type qualifiers and allowing a wide range of approximate code modifications. It was evaluated with PARSEC benchmark on three platforms: a normal PC, an FPGA-augmented mobile SoC, and an energy-harvesting sensor device, with average speedups of 2.3x, 4.8x, and 1.5x respectively.

3.3 HARDWARE APPROXIMATION TECHNIQUES

Hardware approximation is achieved either using faulty hardware systems or systems with timing violations, which operate in voltage or frequency[24] scaled regions. The term "faulty hardware" here refers to introducing inaccuracies in the hardware design rather than actual flaws. [61] presents an approximation floating-point adder based on an inexact mantissa adder and exponent subtractor, with results indicating that the suggested adder's power consumption and delay are reduced by 37% and 62%, respectively, when compared to a single-precision FP adder.

Research work in approximate hardware circuit design has been mostly targeted towards arithmetic circuit level approximations. [38] presents the simulation-based evaluation of approximate adders in image processing application being subjected to voltage over scaling. As per the evaluation, an inexact full adder can function at a lower supply voltage, roughly 0.21v less than an exact full adder with the same number of errors and 30 percent reduced energy use. One of the interesting findings by the paper is that the increase in energy variation due to frequency shift for exact adder is substantially bigger than for voltage-overscaled adder.

An another research paper [52] is based on transistor dynamic variability, using voltage-scaling on circuit and image compression on application level. When compared to running the image compression application at nominal deterministic value, compromises in logic gate circuit design resulted in up to 90 percent energy savings.

[92] proposes an error-tolerant adder that works by separating the input operands whereas a logic complexity reduction strategy based on relaxing numerical precision is put forward in [32]. When compared to previous implementations(using voltage overscaling), the design of approximation multi-bit adders for image and video compression applications shows power savings of up to 60% and area savings of up to 37% with no discernible compromise in output quality[32].

The above-mentioned techniques , however, generally did not consider the significance of processing data for image processing applications as proposed in [11]. The technique is based on the fact that images have informational value that can be used to achieve energy minimization while maintaining performance constraints.

To demonstrate the optimization opportunities for approximate hardware computing along with query processing, [34] sketched out the first idea of ApproxiDB[34], which runs a hybrid machine consisting of both approximate hardware and precise hardware. Physical design, query pro-

cessing, and multi-level optimization issues, as well as future research objectives, are discussed in relation to the implementation of a hybrid machine in the paper.

At a single level, we classified the approximation computing approaches as hardware or software AC techniques. Table 2 summarizes the taxanomy of the aforementioned AC techniques. Several researchers have also proposed and grouped these approaches into diverse groups. Paper[56] characterized the techniques on the practical concerns whereas categories like approximate instruction processing, cloud approximate computing has been put forward in [37].

3.4 CROSS-LAYER AC TECHNIQUES

Evidently, there has been a lot of research and literature on approximation at the software and hardware level, but there has been less focus on when they are applied at multitude levels. Optimization of one layer of the system stack has contributed to power savings and high processing times, however, applying approximate computing for multiple layers of the system stacks can lead to greater gains.

The concept of scalable effort design is introduced by [19] based on the identifying mechanisms at each level of design abstraction, namley circuit, architecture, and algorithm. It demonstrated that the cross-layer optimization improves the system performance, saving 1.42X-2X energy on average in comparison to when approximate techniques implemented at the single stage. The compounded benefits from combining the approximate computing techniques has also been successfully demonstrated in the results by [1]. Multiple stack layers of the system has been changed across a set of different applications from PERFECT suite. The results showed not only a reduction in execution time when relaxed sychronization and loop perforation were used, as well as energy savings when computing with fewer bits, but also the advantages of combining these techniques.

According to paper[72], approximation computing, when applied at all levels of hardware and software, can aid in the resolution of issues with commonly used power management strategies. The study[72] examines some of the major issues related to cross-layer implementation, such as transferring output quality constraints, and provides two open source libraries to help bridge the gap between layers.

[30] proposes using hardware-supported approximation as a defence technique for CNN implementation against adversarial attacks. For a series

Table 2: Taxonomy of AC techniques

Techniques	Related Work	Advantages	Disadvantages
Loop Perforation/ Task Skipping	[75],[53], [55], [15]	dramatically increase performance and energy savings	not appropriate for all applications
Memoization	[4],[23], [44]	technique is machine-independent, works during runtime	only works on acessible functions, data is really poor in case of significant losses.
Voltage Scaling	[24], [38], [52]	widely applicable to a broad range of subsystems, including logic, memory, and even sensors	manufacturing limits, need for level shifters
Precision Downgrade	[17], [36]	reduce some floating point operations to ”trivial” ones, help other techniques such as memoization to work more efficiently, need simple modifications	not much significant savings
Programming language/compilers	[63],[71],[70]	user does not need to be aware of algorithm-specific settings.	can lead to choice of poor quality algorithms and errors
Inexact/Faulty hardware	[61],[32],[92]	highly effective AC technique	not always suitable mainly because they require physical modifications to the components of the system, generally require special architecture

of attacks, experiments were conducted for cross-layer approximate implementation from gate-level up to system-level for the two datasets MNIST and CIFAR-10. The proposed implementation boosts the robustness of a LeNet-5 and an Alexnet CNNs by up to 99 percent and 87 percent, respectively while also saving up to 50 percent in energy usage, according to the empirical data presented in the paper.

Cross-layer approximate computing systems can work effectively without harnessing much power. Software and hardware work in tandem according to an architectural standard in cross-layer approximation computing. ProACt is such a designed hardware system in terms of an architectural specification on the appropriate error criteria which the software that runs the application is aware of and actively leverages such hardware characteristics[14].

However, while these studies show how multi-layer optimization might enhance performance and save energy, they don't show how the advantages are shared across different levels. In other words, they didn't present any models for estimating the energy savings that can be achieved by combining various levels of approximation computing techniques.

3.5 POWER-ACCURACY MODELS

Depending on the target system, power consumption usually consists of several components like base costs depending on the executed instruction, costs of memory accesses, etc. These values form the energy model. Various energy models have been developed to estimate power, which may then be utilized to improve system design.

Without understanding the overall system behaviors in sufficient detail, component level power modeling models hardware components such as CPU, memory hierarchy, and disk subsystem, and quantify the power behavior of both application software and operating system. A few component-based power estimation approaches have been developed, including the spreadsheet model. The component's power dissipation is retrieved from the data sheets and entered into the spreadsheet, and the overall power is calculated by simply adding all of the components together. Powerplay tool[54] is a web-based spreadsheet model that offers a library of power models at several levels of accuracy. Because the interaction among components is unknown and many low-level power optimizations is programmed, therefore this method is not very precise.

[64] proposes a system-call based power modelling approach that uses Finite State Machines (FSM) to model the power states and state transitions

of each component as well as the entire device to solve this challenge. However, when designing FSMs for each component, only three power states are frequently utilised to represent the multiple running states, which remains a constraint.

Instruction-level [5] power analysis techniques estimate the total energy cost of a program by adding the energy consumed while executing instructions of a program. [84] has demonstrated an energy characterisation of ARM CortexA7 and Cortex-A15 CPUs at the instruction level. The ARM instruction set is examined in detail, and the instructions are divided into groups with comparable semantics. A basic linear energy model that takes into consideration two criteria for calculating a program's energy consumption is built as a result of this characterization.

In [6], a tool that receives and profiles the energy consumption of the program on an AT91SAM7x256 microcontroller, in addition to presenting a simple model and validating it against actual hardware with less than 6% error, was built. Wattch[10], an another power estimation tool based on instruction-level modeling at the architectural level is used for analyzing and optimizing power dissipation in microprocessors.

A regression analysis based instruction model for RISC-V processor is presented in [50] which accurately estimate the energy consumption of random instruction sequences with an average error of 2.5%. It should be noted that this model can be extended to other processors as well. [21] proposed a novel data fitting techniques and showcases the advantages of using a nonlinear model to estimate power consumption over the widely used linear regression models.

In order to quantify the effects of approximation techniques, a tool called REACT [89] is created, which combines an application profiler with an energy model and an error injection framework to measure the effects of approximation on applications. REACT derives an analytical, linear model for energy consumption using a machine-learning approach that can be rapidly changed to reflect any approximation techniques. Fig.13 shows a table that presents a taxonomy of selected approximation techniques, shows how REACT's energy model captures their effect, and provides a high-level description of the error model.

A theoretical model to efficiently estimate the error characteristics of approximate adders is presented in [51]. Bayesian Network modeling, a probablistic approach is used in [82] to analyze the impact of precision scaling techniques on accuracy. The method is built on the characterization of a library of approximation operators that are used to develop a Bayesian model to predict how precision scaling affect computation, and it has been

Technique	Energy Model Category	Error Model Description
DRAM Refresh Rate [1, 6]	$\text{Memory}_{\text{St}}$	Last-Access Dependent Bit Flip
Drowsy Cache [3]	$\text{Memory}_{\text{St}}$	Read Upset/Write Failure Bit Error
Load Value Approximation [8]	Memory Access Energy	Load Value Predictor Model
Neural Acceleration [9]	$\text{Compute}_{\text{Dyn/St}}$ & $\text{Memory}_{\text{Dyn}}$	Per-Invocation Random Error
Reduced-Precision FPU [13]	FP Arithmetic Instructions	Narrow Mantissa Floating Point
Underdesigned Multiplier [4]	Int Multiply Instructions	Per-Invocation Random Error
Voltage Overscaling (ALU) [10]	Int Arithmetic Instructions	Random Bit Flips
Precision Scaling (ALU) [14]	Int Arithmetic Instructions	LSB Zeroing

Figure 13: REACT(selected approximation techniques taxonomy)

validated using a collection of relevant software case studies. This allows for fast predictions to be made at design time, allowing informed decisions to be made involving precision scaling approximation technique.

4

IMPLEMENTATION

This chapter explores setting up RISC-V toolchain and simulation for our research work. This includes building RISCV on spike, booting a simple application on it using the PK kernel, and compiling a cross-compiled Linux on the emulation platform. We top it by extending toolchain and Spike simulator with our approximate instructions. All the tools have been compiled from scratch and a few issues faced during implementation have also been addressed.

4.1 EXPERIMENTAL TESTBED

4.1.1 *Building RISC-V tool chain, Spike and Proxy Kernel*

RISC-V software tool chain easily create assembly instructions and sequences. The RISC-V software tool chain includes open-source compilers (e.g., GNU/GCC and LLVM), a full Linux port, a GNU/GDB debugger, verification tools, and simulators. It is basically a standard GNU cross compiler toolchain ported for RISC-V. We use riscv-gcc to compile, assemble, and link your source files. Behaviour of riscv-gcc is similar to standard gcc, except that it produces binaries encoded in the RISC-V instruction set. These compiled binaries can be run on spike, the RISC-V ISA simulator. They can also be used to generate a hexadecimal list of machine code instructions that can be loaded into the instruction memory of a simulated (or real) processor.

To get started, RISC-V toolchain, spike and proxy kernel tools are a requirement. The following steps are performed on an Ubuntu 18.04 machine and closely follow the documentation available on the Github repositories. A 64-bit RISC-V processor is the used for experimentation in this thesis work.

To simply receive all the tools at once, the riscv-tools git repository is cloned along with its submodules. To make them all, the installation path is set and build.sh is run using these commands.

```
git clone https://github.com/riscv/riscv-tools.git
git submodule update --init --recursive
export RISCV=~/riscv
./build.sh
```

Several standard packages are needed prior to building the toolchain. Executing the following command will install those packages.

```
$sudo apt-get install autoconf automake autotools-dev curl python3
libmpc-dev libmpfr-dev libgmp-dev gawk build-essential bison flex
texinfo gperf libtool patchutils bc zlib1g-dev libexpat-dev
```

However, the tools repository is not actively maintained and it results in the error shown in Listing 1. As a result, each tool had to be compiled manually.

```
Building project riscv-isa-sim
../spike_main/disasm.cc: In constructor 'disassmebler_t::disassembler_t(int)':
../spike_main/disasm.cc:275:1: note: variable tracking size limit exceeded with -fvar-tracking-
    assignments, retrying without
disassembler_t::disassembler_t(int xlen)
^~~~~~~~~~~~~~
Installing project riscv-isa-sim
mkdir /home/embedded-systems/Desktop/include
mkdir /home/embedded-systems/Desktop/include/fesvr
mkdir /home/embedded-systems/Desktop/lib
mkdir /home/embedded-systems/Desktop/lib/pkgconfig

Configuring project riscv-pk
Building project riscv-pk
gcc: error: unrecognized argument in option '-mcmodel=medany'
gcc: note : valid arguments to '-mcmodel=' are: 32 kernel large medium small; did u mean 'medium'?
make: ***[file.o] Error 1
```

Listing 1: RISCV error(riscv-tools)

1. Clone the repos for the RISCV GNU toolchain, proxy kernel (pk), and Spike.

   ```
   git clone --recursive https://github.com/riscv/riscv-gnu-toolchain
   git clone https://github.com/riscv/riscv-pk
   git clone https://github.com/riscv/riscv-isa-sim
   ```

2. Set the environment variables.

   ```
   $export RISCV=/path/from/home/to/RISCV
   $export PATH = $PATH:$RISCV/bin
   ```

3. The build is configured in a separate sub directory to produce a toolchain for a 64-bit RISC-V core (RV64GC).

   ```
   $cd riscv-gnu-toolchain
   $mkdir build
   $cd build
   ```

   The command below generates a 64-bit RISC-V core toolchain. By incluing the —with-arch=rv32i option from the command, a 32-bit

core toolchain will be generated instead. Passing the option –enable-multilib to build the toolchain will enable both 32-bit and 64-bit support.

```
$../configure --prefix=\$RISCV
$make
```

'make' command builds a newlib cross-compiler toolchain whereas running 'make linux' command will create RISC-V core toolchain with linux cross-compiler. The C-runtime library used by linux cross-compiler is glibc.

Now, to configure and build for RISC-V Proxy Kernel:

```
$cd ../riscv-pk
$mkdir build
$cd build
$../configure --prefix=\$RISCV --host=riscv64-unknown-elf
$make
$make install
```

Alternatively, the Linux toolchain is used to build this package, by setting –host=riscv64-unknown-linux-gnu. Also, to built 32-bit (RV32) versions, add –with-arch=rv32i flag to the configure command. Finally to build Spike:

```
$cd ../riscv-isa-sim
$mkdir build
$cd build
$../configure --prefix=\$RISCV
$make
$make install
```

```
embedded-systems@embeddedsystems-Precision-3541:~$ riscv64-unknown-elf-gcc helloworld.c -o hello
embedded-systems@embeddedsystems-Precision-3541:~$ /home/embedded-systems/riscv-isa-sim/build/
    spike /home/embedded-systems/riscv-pk/build/pk hello
bbl loader
Hello, World!
```

Listing 2: Test run(helloworld)

A successful hello-world C program test (Listing 2)shows that the compiled tools are working well in proxy emulation. However, there are a few more steps to be completed before RISC-V spike can run in full-linux OS mode.

1. Compiling the linux kernel

```
git clone https://github.com/torvalds/linux.git
cd linux
make ARCH=riscv CROSS_COMPILE=riscv64-unknown-linux-gnu- defconfig
make ARCH=riscv CROSS_COMPILE=riscv64-unknown-linux-gnu- -j $(
    nproc)
```

These commands build the linux kernel in the default configuration. 'make ARCH=riscv menuconfig' can be used to edit the configuration. At the end of the build, a bootable RISC-V kernel can be found under linux/arch/riscv/boot/vmlinux.

2. Building Busybox

   BusyBox is a software suite that provides several Unix utilities in a single executable file. It is eventually able to provide a shell at boot. Menuconfig is used to set the build to static. It can be built dynamically, however this necessitates the copying of numerous libraries.

```
git clone https://git.busybox.net/busybox
cd busybox
make defconfig CROSS_COMPILE=riscv64-unknown-linux-gnu- make
    defconfig
make menuconfig CROSS_COMPILE=riscv64-unknown-linux-gnu- make
    menuconfig
make -j $(nproc) CROSS_COMPILE=riscv64-unknown-linux-gnu-
```

3. Creating rootFS

   The root filesystem is the top-level directory of the filesystem which contains the files and directories critical for system operation, including the device directory and programs for booting the system. After the system is booted, all other filesystems are mounted as subdirectories of the root filesystem on standard, well-defined mount points. Root file system includes directories such as init files when Linux boots. There are shareable, read-only files, including executable binaries and libraries in /usr, the user executable files in /bin, the device files for every hardware device attached to the system and so on. When Linux starts up, the first thing that must be mounted is the root file system; if the system cannot mount the root file system from the specified device, the system will fail and exit the boot. After success, other file systems can be mounted automatically or manually.

```
mkdir root
cd root
```

```
mkdir -p bin etc dev lib proc sbin sys tmp usr usr/bin usr/lib usr
    /sbin
cp <busy-box-path>/busybox bin/busybox
ln -s ../bin/busybox sbin/init
ln -s sbin/init init
sudo mknod dev/console c 5 1
```

In this folder, a file named etc/inittab with the following content is added:

```
::sysinit:/bin/busybox mount -t proc proc /proc
::sysinit:/bin/busybox mount -t tmpfs tmpfs /tmp
::sysinit:/bin/busybox mount -o remount,rw /dev/htifblk0 /
::sysinit:/bin/busybox --install -s
#::respawn:/bin/busybox getty 38400 ttySBI0
/dev/console::sysinit:-/bin/ash
```

The command below create a cpio archive:

```
find . | cpio --quiet -o -H newc > <linux-repo>/rootfs.cpio
```

Linux is configured to embed the cpio file within the vmlinux. This is done using the menuconfig option:

```
make ARCH=riscv menuconfig
```

Under General Setup, "Initial RAM filesystem and RAM disk" is marked. Then the option "Initramfs source file" is selected and "rootfs.cpio" is added. Atlast, exited all the way back and saved to .config. vmlinux is rebuild by "make -j4 ARCH=riscv vmlinux".

4. Rebuilding bootloader

   Before running Spike in full system mode, it is important to clean and rebuild bootloader, bbl.

   ```
   cd riscv-pk
   mkdir build
   cd build
   ../configure --prefix=\$RISCV --host=riscv64-unknown-elf --with-
       payload=path/to/riscv-linux/vmlinux make
   ```

   Troubleshooting:

   There may be a kernel panic error(Listing 3) when running spike in full system mode if any of the procedures listed above are not completed correctly. It was necessary to ensure that the linux kernel and busybox versions used were neither outdated or unstable.

```
[ 0.020000] console [sbi_console0] enabled
[ 0.020000] futex hash table entries: 256 (order: 0, 6144 bytes)
[ 0.020000] workingset: timestamp_bits=61 max_order=19 bucket_order=0
[ 0.030000] jitterentropy: Initialization failed with host not compliant with requirements
    : 2
[ 0.030000] io scheduler noop registered
[ 0.030000] io scheduler cfq registered (default)
[ 0.040000] VFS: Cannot open root device "(null)" or unknown-block(0,0): error -6
[ 0.040000] Please append a correct "root=" boot option; here are the available partitions
    :
[ 0.040000] Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0)
[ 0.040000] CPU: 0 PID: 1 Comm: swapper Not tainted 4.6.2-00048-g9079be6 #1
[ 0.040000] Call Trace:
[ 0.040000] [] walk_stackframe+0x0/0xc8
[ 0.040000] [] panic+0xec/0x20c
[ 0.040000] [] mount_block_root+0x248/0x328
[ 0.040000] [] ksysfs_init+0x10/0x40
[ 0.040000] [] prepare_namespace+0x148/0x198
[ 0.040000] [] kernel_init_freeable+0x1c8/0x200
[ 0.040000] [] rest_init+0x80/0x84
[ 0.040000] [] kernel_init+0x10/0x11c
[ 0.040000] [] rest_init+0x80/0x84
[ 0.040000] [] ret_from_syscall+0x10/0x14
[ 0.040000] ---[ end Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-
    block(0,0)
```

Listing 3: Kernel Panic Error

Also since the new ISA riscv-linux has removed support of HTIF, there is no block device support anymore and the +disk argument(as mentioned in several tutorials), while rebuilding the kernel will not work to specify a root disk image. Therefore, the boot has to be using an initramfs ramdisk. The creation of root filesystem and creating root.cpio file, specifying the location under linux menuconfig resolves the issue.

The issues page of riscv is quite active and helpful in getting rid of errors experienced during getting riscv system working.

4.1.2 *Extending RISC-V with approximate instructions*

A crucial part of our work includes implementation of approximate computing at the instruction level. The foremost step is to introduce new approximate instructions in such a way that the behaviour of the program is unaffected. As a result, it's important to figure out which instructions in a given application program can be approximated. For a graphic applications, like Sobel filter application, the insight is to approximate lower-order bits of the data. For any application, the piece of code that takes most of time or consumes most energy is the most approximable region[91]. In the

case of Sobel filter, addition and multiplication instructions are executed multiple times. Therefore in this work, arithmetic instructions i.e., multiplication and addition are selected for approximation.

We augmented RISCV processor with approximate addition and multiplication operators. In order to do so, new custom instructions are added to our RISC-V architecture with the riscv-gnu-toolchain. One way to achieve this is by initializing the instruction in riscv-opc.c file. The opcode and opcode mask of the custom instructions is required for this purpose. To obtain these, riscv-tools repository needs to be cloned. Running a simple git command will serve the purpose.

```
git clone https://github.com/riscv/riscv-tools.git
```

The opcodes and instruction bits assigned to different instructions can be found in riscv-tools/riscv-opcodes/opcodes file. For the approximate instruction, the instruction bits assigned with the values are added to the file. The opcodes for approxadd and approxmul, as shown in Listing 4, correspond to the custom-0 and the custom-1 values. The values bits[6 : 2] are set equal to 00010 and the bits[1 : 0] are set to 1 for both approxadd and approxmul instructions. These values are used to ensure that there are no overlaps. The opcode values for the new instructions can be anything we want as long as they conform to one of the RISC-V instruction formats, but the values cannot be the same for two instructions. Opcodes tagged custom-0 and custom-1 are ignored by standard extensions and are recommended for usage within the RISC-V instruction set format for custom instructions[86].

```
# format of a line in this file:
# <instruction name> <args> <opcode>
# <opcode> is given by specifying one or more range/value pairs:
# hi..lo=value or bit=value or arg=value (e.g. 6..2=0x45 10=1 rd=0)
# <args> is one of rd, rs1, rs2, rs3, imm20, imm12, imm12lo, imm12hi,
# shamtw, shamt, rm
ori     rd rs1 imm12           14..12=6 6..2=0x04 1..0=3
andi    rd rs1 imm12           14..12=7 6..2=0x04 1..0=3

approxadd rd rs1 rs2 31..25=0  14..12=7 6..2=0x02 1..0=3
approxmul rd rs1 rs2 31..25=2  14..12=7 6..2=0x02 1..0=3
add       rd rs1 rs2 31..25=0  14..12=0 6..2=0x0C 1..0=3
sub       rd rs1 rs2 31..25=32 14..12=0 6..2=0x0C 1..0=3
sll       rd rs1 rs2 31..25=0  14..12=1 6..2=0x0C 1..0=3
```

Listing 4: RISC-V opcodes

Then, the following command is used to obtain the match and mask opcode for the custom instruction:

```
cat opcodes-pseudo opcodes opcodes-rvc opcodes-rvc-pseudo opcodes-
    custom | ./parse-opcodes -c > <path name>/temp.h
```

The opcode from the temp file is copied to riscv-opc.h(Listing 5), located at riscv-gnu-toolchain/riscv-gdb/include/opcode/riscv-opc.h.

```
#ifndef RISCV_ENCODING_H
#define RISCV_ENCODING_H
#define MATCH_APPROXADD 0x700b
#define MASK_APPROXADD  0xfe00707f
#define MATCH_APPROXMUL 0x400700b
#define MASK_APPROXMUL  0xfe00707f
#define MATCH_ADD 0x33
#define MASK_ADD  0xfe00707f
#define MATCH_SUB 0x40000033
#define MASK_SUB  0xfe00707f

#endif
#ifdef DECLARE_INSN
DECLARE_INSN(approxadd, MATCH_APPROXADD, MASK_APPROXADD)
DECLARE_INSN(approxmul, MATCH_APPROXMUL, MASK_APPROXMUL)
DECLARE_INSN(add, MATCH_ADD, MASK_ADD)
DECLARE_INSN(sub, MATCH_SUB, MASK_SUB)
#endif
```

Listing 5: RISC-V match/mask opcodes

And then the riscv-gnu-toolchain/riscv-gdb/opcodes/riscv-opc.c (Listing 6) file is edited. The constructors here use a number of parameters, such as d for destination, s for source.

```
const struct riscv_opcode riscv_opcodes[] =
{/* name, xlen, isa,   operands, match, mask, match_func, pinfo.  */
{"add",         0, INSN_CLASS_I,   "d,s,t",  MATCH_ADD, MASK_ADD,
    match_opcode, 0 },
{"approx_add",         0, INSN_CLASS_I,   "d,s,t",  MATCH_APPROXADD,
    MASK_APPROXADD, match_opcode, 0 },
{"mul",       0, INSN_CLASS_M,   "d,s,t",  MATCH_MUL, MASK_MUL,
    match_opcode, 0 },
{"approxmul",       0, INSN_CLASS_M,   "d,s,t",  MATCH_APPROXMUL,
    MASK_APPROXMUL, match_opcode, 0 },
```

Listing 6: riscv-opc.c

Atlast, the toolchain is compiled again using the make command. The outlined steps above are simple to follow, and any additional custom instructions can be readily added. However, every time we add a new instruction, the process of building the toolchain takes a long time. Therefore, an another method to add custom instructions is used. The pseudo assembly directive .insn provided by riscv-binutils allows us to insert our approximate custom instructions. This directive permits the numeric representation of instructions and makes the assembler insert the operands according to one of the instruction formats for '.insn'. For example, the instruction 'add a0, a1, a2' could be written as '.insn r 0x33, 0, 0, a0, a1, a2'. The compiler files the register fields itself according to our operands.

For approximate instructions, R-type instruction is chosen because multiplication and addition are basically a normal arithmetic operation. Since we are using custom-0 and custom-1 opcodes for the implementation, funct3 field is zero in case of approxmul and it is decimal 2 in case of approxadd. funct7 is same for both approximate custom instructions. The numbers are entered and the instruction now should be recognized by the riscv-gnu-toolchain.

The C program (Listing 7) with this instruction is compiled to test and verified the functionality of the added instructions. After adding the approximate instructions to the simulator, the output of the program is displayed.

```
#include <stdio.h>
int add(int res, int op1, int op2){
        asm __volatile(".insn r 0x0B, 0x7, 2, %0, %1, %2"
        : "=r" (res) : "r" (op1), "r" (op2)\
        ); // assembly statement for approxadd
        return res;
}
int mul(int res, int op1, int op2){
        asm __volatile(".insn r 0x0B, 0x7, 0, %0, %1, %2"
        : "=r" (res) : "r" (op1), "r" (op2)
        ); // assembly statement for approxmul
        return res;
}
int main(int argc, char** argv) {
        int a = 0, b = 11125, c = 11125; //random numbers for testing
        printf("The sum of two numbers is %d\n", add(a, b, c));
        printf("The result of multiplication is %d\n", mul(a, b, c));
}
```

Listing 7: A simple C test program

4.1.3 *Extending Spike Simulator*

We extended the vanilla Spike simulator in two ways: (1) we added tracing capabilities to count the number of instructions executed during out test scenario; and (2) we added approximate versions of addition and multiplication (trimming least significant bits of the operands and the result).

The ability to print the number of times a single instruction is executed is also aided by the enhanced tracing capabilities. This functionality came in handy to figure out which architectural instructions were to be approximated and when determining how many approximate instructions were executed. It should be noted that all instructions are executed in a single cycle, although the number of cycles per instruction can be easily modified.

```
embedded-systems@embeddedsystems-Precision-3541:~$ /home/embedded-systems/riscv-isa-sim/build/
     spike -l /home/embedded-systems/riscv-pk/build/pk addition
bbl loader
The sum of the numbers is 50

Number of instructions executed:384252
add insn count:94572
```

Listing 8: RISC-V instruction count

In order to include approximate versions of instructions in Spike, certain files are added/modified.

1. The riscv/insns/<approxinstructionname>.h file defines the functional behaviour of the instructions in Spike, which is why the approxadd.h and approxmul.h files are added to determine the behaviour of approximate instructions.

```
reg_t rs1 = (RS1 >> (6))<<(6);
reg_t rs2 = (RS2 >> (6))<<(6);
WRITE_RD(sext_xlen((rs1+rs2)));
```

Listing 9: approxadd.h

```
reg_t rs1 = (RS1 >> 6)<<6;
reg_t rs2 = (RS2 >> 6)<<6;
WRITE_RD(sext_xlen((rs1*rs2)));
```

Listing 10: approxmul.h

2. The riscv/encoding.h file declares all the instructions. As a result, the custom instructions, namely, approxadd and approxmul, as well as their opcode and opcode masks, are added to this file. The opcodes

and opcode masks are the same as used in the toolchain (obtained using riscv-tools).

3. Finally, the instructions are added to riscvinsn list, i.e. riscv.mk.in file and then simulator is rebuilt.

```
riscv_gen_hdrs = \
        icache.h \
        insn_list.h \

riscv_insn_ext_i = \
        add \
        approxadd \
        approxmul \
        addi \
        addiw \
        addw \
        and \
```

Listing 11: Spike instruction list(riscv.mk.in file)

The functionality of the simple addition and multiplication software utilising approximation assembly instructions(Listing 7) was verified, and the output was correctly displayed with some error(Listing 12), as expected, since the approximate computing technique bit-width reduction is implemented at the architectural level.

```
embedded-systems@embeddedsystems-Precision-3541:~$ riscv64-unknown-elf-gcc insn.c -o insn -lm
embedded-systems@embeddedsystems-Precision-3541:~$ /home/embedded-systems/riscv-isa-sim/build/
    spike -l /home/embedded-systems/riscv-pk/build/pk insn
bbl loader
The sum of the numbers is 22144
The result of multiplication is 122589184
```

Listing 12: Test run Spike for approximate instructions

4.2 MULTI-LAYER APPROXIMATION MODEL

Our work focuses on modeling how knowledge of the impact of approximations at each layer can be composed to predict (and thus guide the design) the impact of multi-layers approximation. I.e., if we know the impact on and accuracy of approximations at layer L_n and layer L_{n+1} (where higher indices represent higher layers), can we predict the impact of approximations on both layers?

We model this impact by assuming a multiplicative effect of energy savings across layers: i.e., let E_0 represent the baseline, un-approximated,energy consumption, and $E\prime$ represent the actual energy consumption after approximations. Let $G_n \in]0,1[$ represent the gain in energy consumption of approximations at layer n, such that, if only approximations at that layer are applied, $E\prime = G_n \times E_0$. By modeling energy reductions as multiplicative (i.e., such that energy gains across layers are orthogonal), we can define $E\prime = (G_M \times G_{M-1} \times G_{M-2} \times ... \times G_1) \times E_0$, for M layers.

Modeling the effects on accuracy is more interesting, as, although these are also multiplicative, they are weighted by effects on upper layers. Let A_0 represent the baseline, un-approximated, accuracy, and $A\prime$ represent the actual accuracy after approximations. Let $R_n \in]0,1[$ represent the reduction in accuracy of approximations at layer n, such that, if only approximations at that layer are applied, $A\prime = (1 - R_n) \times A_0$. If an approximation at layer L_{n+1} has already been applied, the effective $R_n\prime$ is modified from the original R_n by $R_n\prime = R_n \times R_{n+1}$. Thus, we can define $A\prime = ((((1-(R_M))-(R_M \times R_{M-1}))-(R_M \times R_{M-1} \times R_{R_M-2}))-...-((R_M \times R_{M-1} \times R_{R_M-2} \times ... \times R_1))) \times A_0$. In other words, the cost of approximations at each layer go down once we have approximated at upper layers.

Designing an approximation strategy for a given energy/accuracy profile, across several layers, can thus be achieved by determining (empirically or analytically) respective G and R for all layers where approximations are possible, and solving the equations governing $E\prime$ and $A\prime$ for the possible combinations of G_n and R_n. Clearly, precise solutions will seldom exist, so we require an strategy that optimizes for the smallest $E\prime$ and highest $A\prime$ (within application constraints); these sorts of strategies are outside the scope of this thesis work.

5

EXPERIMENTS AND RESULTS

The experimental setup for collecting empirical results is shown in Fig.14. For a representative image processing application, standard test images from the University of Southern California database are used. The application is complied with the riscv-gcc compiler and runs on the Spike Simulator. Approximations are employed on two levels. Loop perforation is utilised in software algorithmic implementation, while arithmetic approximation instructions are employed in hardware. Spike simulator supports hardware approximation when extended with approximate addition and multiplication instructions. Energy results are computed using a power model as Spike outputs the instruction count(cycles per instruction(cpi) is 1). Approximate computing trades accuracy for energy savings but there is fundamental limit to which the loss in accuracy is allowed. Therefore, the quality of results produced further needs to be inspected and assessed using quality metrics.

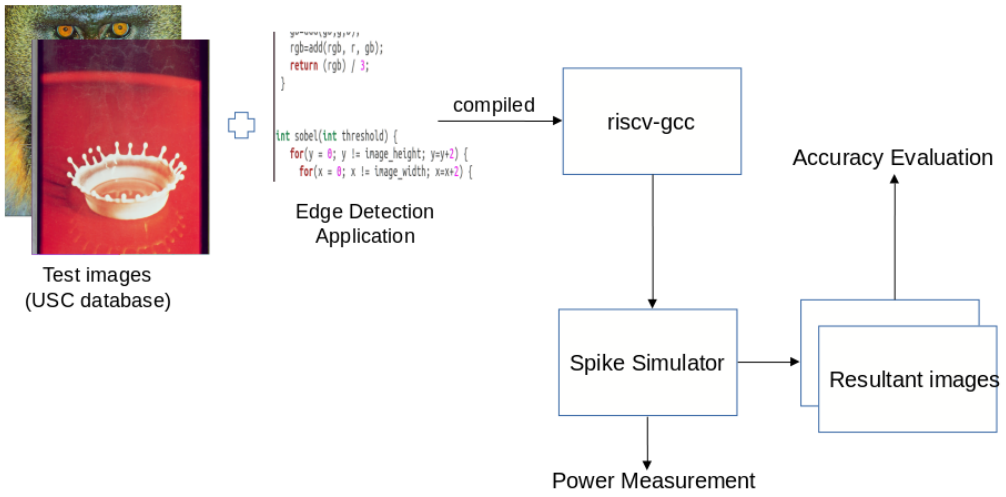


Figure 14: Experimental Setup

5.1 SOBEL FILTER APPLICATION AND TEST IMAGES

A Sobel filter is used, accepting as input a bitmap image and finding its edges. The output is a grayscale image, with pixel values ranging from the value 0 to 255. A 3x3 kernel matrix is applied in both x and y direction

(horizontal and vertical gradient). These two gradient values are squared, added and the square root of the sum provides the magnitude of the gradient. Pixels that have gradients of large magnitude are likely part of an edge in an image . For the purposes of testing, 15 standard test images from the University of Southern California database[80] are used. Listing 13 shows how application program uses inline assembly approximation instructions to accomplish bit-width reduction for addition and multiplication operations.

```
#include "bitmap.h"

int mask[2][3][3] = {
        {{-1,-2,-1},
                {0 , 0, 0},
                {1 , 2, 1}},

        {{-1, 0, 1},
                {-2, 0, 2},
                {-1, 0, 1}}
};

int gb,rgb=0;

int add(int a, int lhs, int rhs){
        asm __volatile(".insn r 0x0B, 0x7,2, %0, %1, %2"
        : "=r" (a)\
        : "r" (lhs), "r" (rhs)\
        );
        return a;
}

int mul(int a, int lhs, int rhs){
        asm __volatile(".insn r 0x0B, 0x7,0, %0, %1, %2"
        : "=r" (a)\
        : "r" (lhs), "r" (rhs)\
        );
        return a;
}

int turngray(int r, int g, int b) {
        gb=add(gb,g,b);
        rgb=add(rgb, r, gb);
        return (rgb) / 3;
```

```
}


int sobel(int threshold) {
        int  x, y, dy,dx, width;
        char R, G, B;
        int sum,sumGx, sumGy, Gx, Gy;
        int filterRadius;

        for(y = 0; y != image_height; y=y+2) {
                for(x = 0; x != image_width; x=x+2) {
                        filterRadius = 1;
                        sumGx=0.0;
                        sumGy=0.0;
                        for(dy = -filterRadius; dy != 2; dy++) {
                                for (dx = -filterRadius; dx != 2; dx++)
                                    {
                                        if (x + dx >= 0 && x + dx<
                                            image_width && y + dy >= 0
                                            && y + dy < image_height)
                                        {
                                                R = *(source_image +
                                                    byte_per_pixel * (
                                                    image_width * (y+dy)
                                                     + (x+dx)) + 2);
                                                G = *(source_image +
                                                    byte_per_pixel * (
                                                    image_width * (y+dy)
                                                     + (x+dx)) + 1);
                                                B = *(source_image +
                                                    byte_per_pixel * (
                                                    image_width * (y+dy)
                                                     + (x+dx)) + 0);
                                                sumGx +=    mul(sumGx,
                                                    turngray(R, G, B),
                                                    mask[0][dx +
                                                    filterRadius][dy +
                                                    filterRadius]);
                                                sumGy +=    mul(sumGy,
                                                    turngray(R, G, B),
                                                    mask[1][dx +
                                                    filterRadius][dy +
                                                    filterRadius]);
```

```
                                        }
                                }
                        }


                sum=0;
                Gx, Gy=0;
                Gx=mul(Gx,sumGx,sumGx);
                Gy=mul(Gy,sumGy,sumGy);
                sum=add(sum,Gx,Gy);
                sum = sqrt(sum);

                if (sum - threshold >= 0) {
                        *(output_image + byte_per_pixel * (image_width
                            * y + x) + 2) = 0;
                        *(output_image + byte_per_pixel * (image_width
                            * y + x) + 1) = 0;
                        *(output_image + byte_per_pixel * (image_width
                            * y + x) + 0) = 0;
                                }
                else {
                        *(output_image + byte_per_pixel * (image_width
                            * y + x) + 2) = 255;
                        *(output_image + byte_per_pixel * (image_width
                            * y + x) + 1) = 255;
                        *(output_image + byte_per_pixel * (image_width
                            * y + x) + 0) = 255;
                                }
                }
        }

        return 0;
}
```

Listing 13: approxsobel.c file

5.2 QUALITY METRICS

To assess the performance benefits from approximate computing techniques, it is necessary to determine whether the results produced are acceptable. To do so, it is important to understand the quality metrics typically used for

these applications, and compare the observed quality with and without using approximate computing techniques.

We evaluate "accuracy" using peak signal-to-noise ratio (PSNR) and structural similarity (SSIM) assess of inexact designs compared to baseline (exact) implementation. PSNR is peak signal-to-noise ratio, in dB between two images. The most straightforward way to define PSNR is via the mean squared error (MSE). The MSE represents the cumulative squared error between the compressed and the original image, whereas PSNR represents a measure of the peak error. The lower the value of MSE, the lower the error and the higher the PSNR, the better the quality of the compressed, or reconstructed image. When two images are the same the MSE will give zero, resulting in an invalid divide by zero operation in the PSNR formula. The transition to a logarithmic scale is made because the pixel values have a very wide dynamic range.

To compute the PSNR, we first calculate the mean-squared error using the following equation:

$$MSE = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \frac{[I1(i,j) - I2(i,j)]^2}{m * n}$$

where I1 is the original image and I2 is the approximate image, m and n are the number of rows and columns in the images.

$$PSNR = 10log_{10}(\frac{R^2}{MSE})$$

where R is the maximum possible pixel value of the image. Since we are using a bitmap image where the pixels are represented using 8 bits per sample, this is 255.

PSNR is the most commonly used metric to measure the quality in the image processing domain. Each PSNR value is calculated in the experiments by comparing the approximated resultant image to the output image obtained by accurately running Sobel Filter application. The PSNR for the accurate image is assumed to be 100 dB since it is used as a reference image. The PSNR measures the absolute errors between pixels of two compared images but is less related to human perception. The SSIM is a metric used for assessing the quality of human perception.

The SSIM index is calculated on various windows of an image. It models any picture distortion as a combination of three factors: luminance distortion(l), loss of correlation(c), and contrast distortion(s). Python scripts are used to determine PSNR and SSIM.

$$SSIM(x,y) = [l(x,y)]^{\alpha} \times [c(x,y)]^{\beta} \times [s(x,y)^{\gamma}]$$

where:

$$l(x,y) = \frac{(2\mu_x\mu_y + C_1)}{(\mu_x^2 + \mu_x^2 + C_1)}$$

$$c(x,y) = \frac{(2\sigma_x\sigma_y + C_2)}{(\sigma_x^2 + \sigma_x^2 + C_2)}$$

$$s(x,y) = \frac{(\sigma_x y + C_3)}{(\sigma_x\sigma_y + C_3)}$$

where μ represents the mean of a given image, σ denotes the standard deviation of a given image, x and y are the two images being compared and C1, C2 is given by:

$$C1 = (K_1 L)^2$$

$$C2 = (K_2 L)^2$$

$$C3 = C2/2$$

and K_1=0.01, K_2=0.03 and L is the dynamic range for pixel values, 255 for standard 8-bit images.

5.3 TEST SCENARIOS

We perform approximations at two layers: hardware and software. At the hardware level (bottom layer), the previously described approximate versions of addition and multiplication instructions are employed to run the test program (unmodified at source level) in an approximate manner. At the software level, we apply a loop perforation technique. Software is implemented in C and compiled using riscv-gcc. The application uses nested loops, so rather than reducing the upper limit of the loop, the step in the iteration is increased as shown in Listing 14.

```
//changing the expression from
for(y=0; y!=height; y++){
        for(x=0;x!=width;x++){
//To:
for(y=0; y!=height; y+2){
        for(x=0;x!=width;x+2){
```

Listing 14: Loop Perforation (rate 50%)

Thus, we can compare 4 different versions: baseline (exact), software-only approximations (loop perforation), hardware-only approximations (approximate arithmetic), and software/hardware approximations.

5.4 POWER MODEL

As our work is thus far only evaluated on a functional simulator, rather than a simulator/implementation that can provide accurate energy consumption results; thus, we assess energy savings based on a simplified power consumption model. We assume one energy unit, 1 Joule(J) is consumed per executed instruction (this model is accurate enough for modern processors assuming CPI is close enough to [28]). For approximate instructions, we assume 0.3 energy units are consumed per instruction execution (aligned with estimates found in literature [73]). As our research progresses to use more realistic power models, results must be revised, but this model suffices at present to validate our methodology.

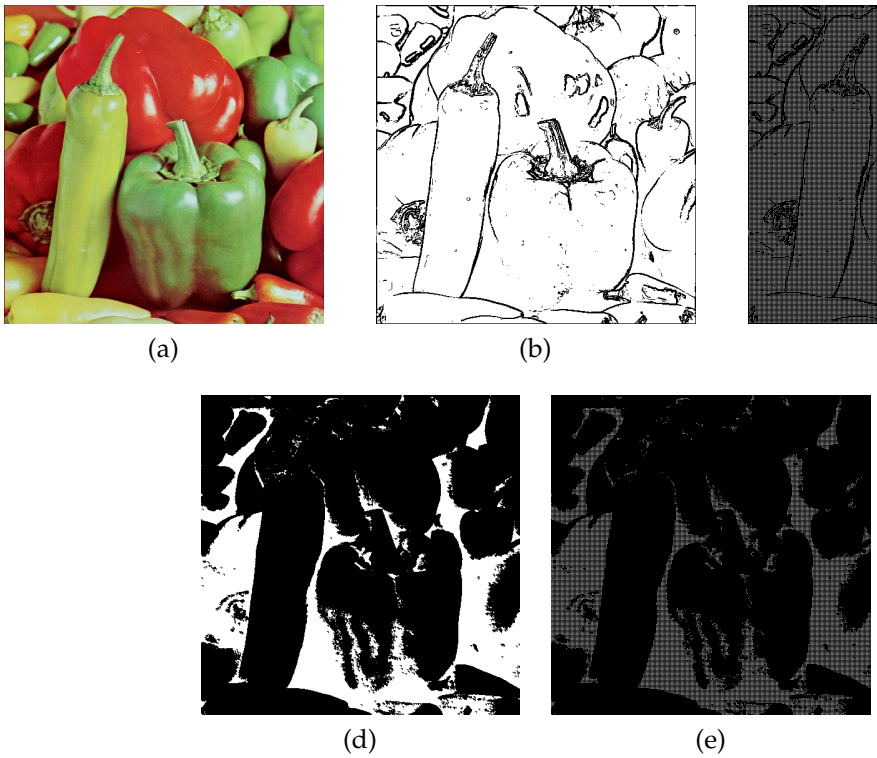
5.5 MODELING APPROXIMATIONS ACROSS LAYERS

As we are modeling two layers, the energy equation previously described to determine energy reduction due to approximations can be precisely written as $E\prime = (G_2 \times G_1) \times E_0$, for 2 layers. The accuracy equation can be precisely written as $A\prime = ()((1 - (R_2)) - (R_2 \times R_1)) \times A_0$, for 2 layers.

We empirically calculate G and R values as the ratio between estimated energy for exact and corresponding approximate versions (for G) and between PSnR (for R).

5.6 EMPIRICAL RESULTS

Fig.15, 16, 17 shows example input images (Fig. 15-a, 16-a, 17-a) and output image of edge detection for: exact execution (Fig. 15-b, 16-b, 17-b); Software approximations, i.e., loop perforation where each loop is perforated by skipping every other execution for a perforation rate of 50 percent (Fig. 15-c, 16-c, 17-c); Hardware approximations, for bit-width reduction where six least-significant bits are truncated (Fig. 15-d, 16-d, 17-d)); and the resultant image when both hardware and software approximation techniques are applied simultaneously(Fig. 15-e, 16-e, 17-e)). Visual inspection reveals that all outputs exhibit correct edges. Where loop perforation is applied, a visible artifact appears (grid lines corresponding to skipped pixels).



(a) (b) (c)

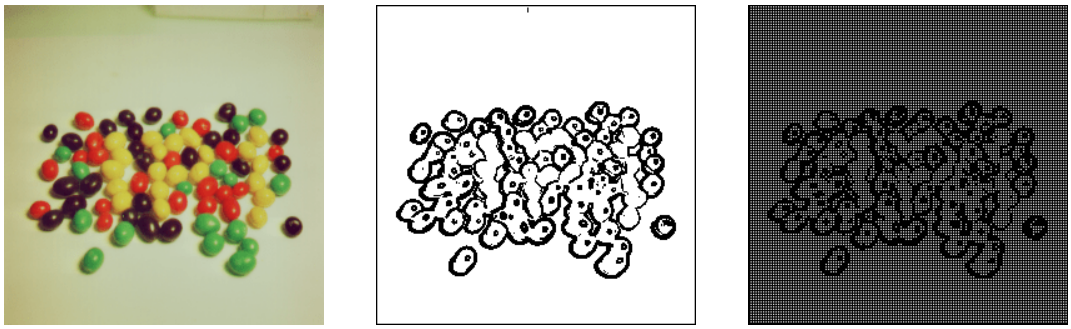
(d) (e)

Figure 15: Example 1 of input image and resultant processed output images for combinations of different approximations.

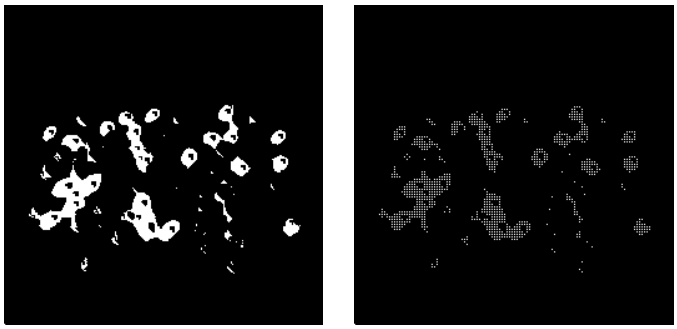
Where approximate arithmetic is applied, shading occurs within edged areas (this is likely because of the sliding window, working on approximate instructions, propagating negatively saturated pixel values between edges, but we have not yet investigated further).

Table 3 indicates the proportion of approximate instructions over the total number of executed instructions when hardware approximation and software-hardware approximation are used. These numbers are obtained using our extended Spike simulator.

Table 4 depicts consumed energy, as per our power model, and Peak Signal to Noise Ratio (PSnR) and SSIM for Fig.14(output images). PSnR and SSIM values are calculated using the native PSnR formula with baseline output image (Fig. 15-b) as the reference image. Table 5 depicts G and R values derived from Table 4, and resulting estimated $E\prime$ and $A\prime$, as predicted by our approximation model, and compared to real evaluation. Fig.18 shows the mean and standard deviation PSnR and SSIM values for the 15 test images. It is evident from the plot that there isn't much variation in the quality of the resultant images.



(a) (b) (c)



(d) (e)

Figure 16: Example 2 of input image and resultant processed output images for combinations of different approximations.

(a) (b) (c)

(d) (e)

Figure 17: Example 3 of input image and resultant processed output images for combinations of different approximations.

Table 3: Approximate/Total Executed Instructions

Version	App. Instr.	Total Instr.	Proportion
Hw. app.	14905368	980217449	0.015
Sw./Hw. app.	3726342	260822205	0.014

Table 4: Empirical Results

Version	Energy consumption(J)	PSnR	SSIM
Baseline	980533055	100	1
Sw. app.	260901237	49.7548728004	0.935
Hw. app.	969783691.4	49.9133568536	0.916
Sw./Hw. app.	258213765.6	48.9122574781	0.899

Table 5: Multi-layer model performance

Parameter	Estimated value	Accuracy (%)
G_2	0.266	-
G_1	0.989	-
$E\prime$	257952753	99.89
R_2	0.502	-
R_1	0.501	-
$A\prime$	24.64	50.37

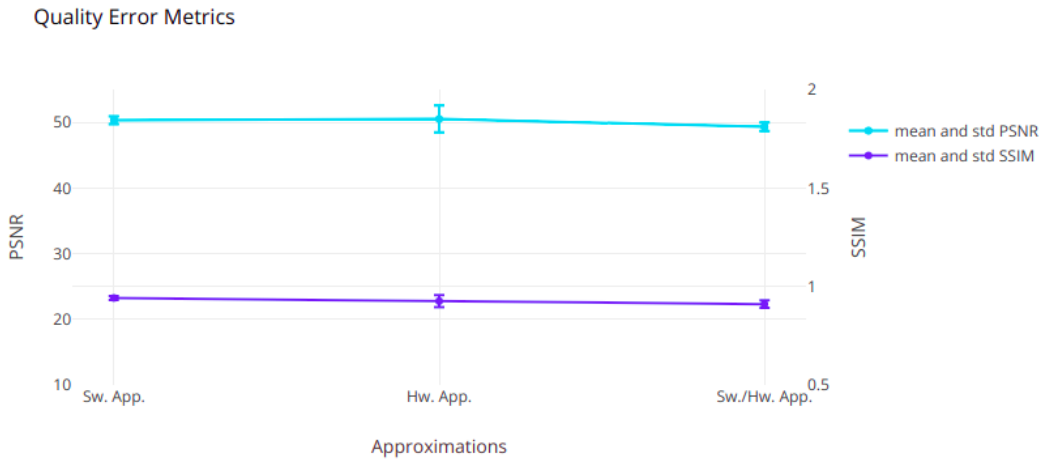


Figure 18: PSNR and SSIM between baseline and approximate images

5.7 REVISING MODEL

Our multi-layer model, for the evaluated test cases, predicts energy consumption with an accuracy of 99%, leading us to believe this multi-layer model successfully captures the behaviour of energy consumption as it is affected by approximations at multiple layers. For accuracy, however, our model is in fact pessimistic: estimating an accuracy of just 24%, when the reality is twice as good. Since the multi-layer multiplicative model(model of type A) seems to be not very successful in estimating accuracy when approximate computing techniques are applied concurrently, it became imperative to revise our initial assumed model. The model clearly required update; evaluation of data is necessary to determine which function best fits empirical results.

The goal is to provide a general model that can provide estimation of energy and accuracy without need to go over the simulation. We analyzed the evaluation results and accordingly made revisions to our model. Big O notation characterizes functions according to their growth rates. A description of a function in terms of big O notation usually provides an upper bound on the growth rate of the function. The mathematical function used in our assumed model is linear for which the accuracy results are pessimatic, implying that a function with a lower growth rate is required. As per orders by rate of growth, the logarithmic function has a growth rate slower than a linear multiplicative function. Therefore, we revised the model and used a logarithmic function model instead.

The revised model estimate the impact by assuming a logarithmic effect of energy savings across layers: i.e., similar to our initial model, let E_0 represent the baseline, un-approximated, energy consumption, and $E\prime$ represent the actual energy consumption after approximations. Let $G_n \in]0,1[$ rep-

Table 6: Revised multi-layer model performance

Parameter	Estimated value	Accuracy (%)
G_2	0.266	-
G_1	0.989	-
$E\prime$	222713936.90	86.25
R_2	0.502	-
R_1	0.501	-
$A\prime$	49.50	98.79

resent the gain in energy consumption of approximations at layer n, such that, if only approximations at that layer are applied, $E\prime = G_n \times E_0$. By modeling energy reductions as logarithmic , we can define $E\prime = \ln(G_M + G_{M-1} + G_{M-2} + ... + G_1) \times E_0$, for M layers.

Again for accuracy, let A_0 represent the baseline, un-approximated, accuracy, and $A\prime$ represent the actual accuracy after approximations. Let $R_n \in]0, 1[$ represent the reduction in accuracy of approximations at layer n, such that, if only approximations at that layer are applied, $A\prime = (1 - R_n) \times A_0$. If an approximation at layer L_{n+1} has already been applied, the effective $R_n\prime$ is modified from the original R_n by $R_n\prime = \ln(R_n + R_{n+1})$. Thus, we can define $A\prime = ((((1 - (R_M)) - \ln(R_M + R_{M-1})) - \ln(R_M + R_{M-1} + R_{R_M-2})) - ... - \ln((R_M + R_{M-1} + R_{R_M-2} + ... + R_1))) \times A_0$.

The equation as per our revised model for energy, therefore can now be written as $E\prime = \ln(G_2 + G_1) \times E_0$, for 2 layers. The accuracy equation can be precisely written as $A\prime = (((1 - (R_2)) - \ln(R_2 + R_1)) \times A_0$, for 2 layers. Table 6 displays the results obtained using logarithmic model.

5.8 DISCUSSION

When cross-layer AC is used, the gain in energy consumption is larger than max(G1, G2), illustrating the benefit of implementing Approximate Computing at multiple levels. There is a limit to how much quality must be compromised in order for the outcomes to be acceptable. There does not appear to be any agreement among researchers in the field of AC on what this number should be, so the quality limit(SSIM) for the images here is presumed to be 0.8, and it can be observed that all images have a greater value, indicating acceptable degradation.

Our initial multi-layer model (type A) accurately estimated the energy consumption to the accuracy of 99 percent, but because the accuracy pre-

diction is only 50% accurate, it is clearly not the appropriate fit for estimating accuracy. The revised model, on the other hand, effectively shows the behaviour of accuracy when it is affected by approximations at multiple layers with a 98% precision but has lower accuracy in terms of energy consumption. Fig. 19 shows the energy and accuracy results of simulation, type-A model and type-B model.

In comparison, the logarithmic model is a better fit for energy and accuracy than the initial model because it is closer to the simulation findings. However, it is observed that unlike energy consumption, both single-layer and cross-layer approximations had a nearly comparable influence on accuracy. The empirical data clearly suggest that since different approximations used at single or multiple levels have varied impact on energy and accuracy, they cannot have the same function models. Using a multiplicative model for predicting energy and a logarithmic model for estimating accuracy would result in a successful multi-layer model (type B), estimating both energy and accuracy within 2% precision as shown in table 7.

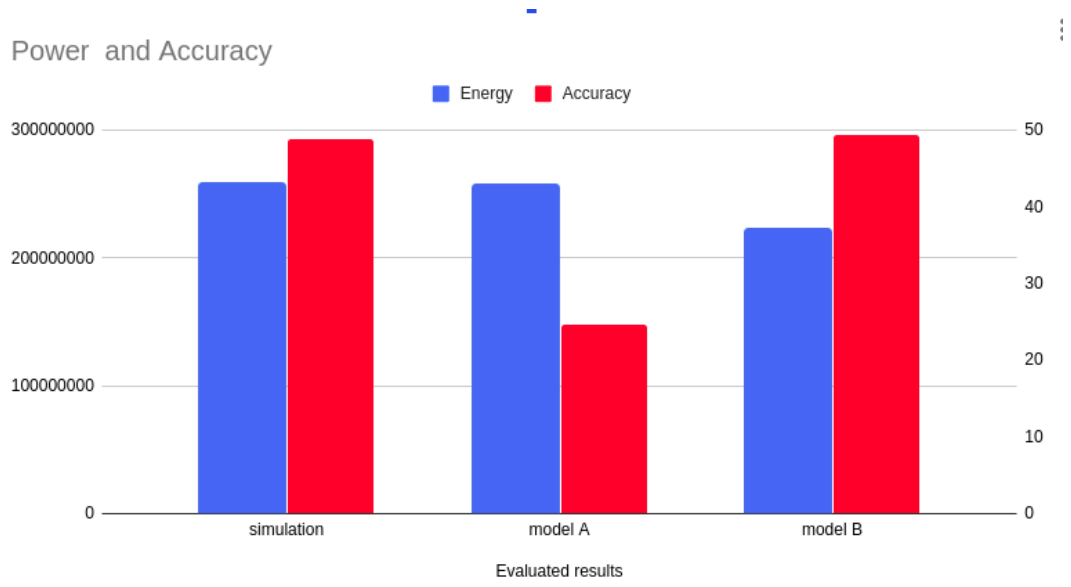


Figure 19: Simulation and Model Results

It should also be noted that different accuracy metrics might also play an important role here: we used PSnR, and perhaps those metrics (signal and noise) embed information that are not adequately processed by our models: other accuracy metrics may be investigated to see which are best modeled (and correlate with energy consumption and accuracy).

Table 7: Combined successful multi-layer model performance

Parameter	Estimated value	Accuracy (%)
G_2	0.266	-
G_1	0.989	-
$E\prime$	257952753	99.89
R_2	0.502	-
R_1	0.501	-
$A\prime$	49.50	98.79

6

CONCLUSION AND FUTURE WORK

This thesis presented a multi-layer model for the effects on energy and accuracy of approximate computing. Our evaluations of an example image processing filter application on a RISC-V simulator assessed output image quality across and estimated energy consumption using a simple power cost model. Results suggest that multi-layer approximations preserve the structural similarity of the output images, suggesting (both formally and visually) that output degradation is acceptable. Empirical results show that the described multiplicative multi-layer approximations model predicts energy savings within a 1% difference, but pessimistically estimates resultant accuracy. The revised model, however, predicted accuracy within 2% difference, but underestimated energy usage. We conclude that a general model can be created by utilising a multiplicative function for energy prediction and a logarithmic function for accuracy prediction. The small discrepancy between the combined model outputs and simulations (less than 2%) demonstrates its success. The combined model can be used efficiently to determine the impact of energy and accuracy on multiple levels.

This is really positive from the point of view of the applicability of multi-layer approximations across more applications with a larger set of comprehensive testing. One of the model's limitations, however, is that it doesn't account for the affect the inputs/outputs from one layer on another. This could be a fascinating subject to investigate. We are aware that the lower level layer's results are passed into the upper level layer, affecting the overall results. However, we have no idea how the two approaches, loop perforation and bit-width reduction, interacted when they are used at the same time.

Furthermore, the experimentation can be expanded to include another approximation computing technique at an another level. To expand the capabilities of the model capabilities, new functional approximation approaches can be included. The model works for loop perforation at the software level and bit-width reduction at the hardware level, but because there are so many different AC approaches, a future direction would be to see how flexible our model is and if it can be extended to other Approximate Computing techniques. The model can be further modeled based on the results gained, resulting in a final general model that can be utilized

by anyone starting work on multi-layer approximate computing without having to rely on simulation-based results.

BIBLIOGRAPHY

[1] A. Agrawal, J. Choi, K. Gopalakrishnan, S. Gupta, R. Nair, J. Oh, D. A. Prener, S. Shukla, V. Srinivasan, and Z. Sura. 'Approximate computing: Challenges and opportunities.' In: *2016 IEEE International Conference on Rebooting Computing (ICRC)*. 2016, pp. 1–8. DOI: 10.1109/ICRC.2016.7738674.

[2] Adel Al-Alawi. 'WiFi Technology: Future Market Challenges and Opportunities.' In: *Journal of Computer Science* 2 (Jan. 2006). DOI: 10.3844/jcssp.2006.13.18.

[3] Vicki H. Allan, Reese B. Jones, Randall M. Lee, and Stephen J. Allan. 'Software Pipelining.' In: *ACM Comput. Surv.* 27.3 (Sept. 1995), 367–432. ISSN: 0360-0300. DOI: 10.1145/212094.212131. URL: https://doi-org.proxy.library.carleton.ca/10.1145/212094.212131.

[4] C. Alvarez, J. Corbal, and M. Valero. 'Fuzzy memoization for floating-point multimedia applications.' In: *IEEE Transactions on Computers* 54.7 (2005), pp. 922–927. DOI: 10.1109/TC.2005.119.

[5] Mostafa Bazzaz, Mohammad Salehi, and Alireza Ejlali. 'An Accurate Instruction-Level Energy Estimation Model and Tool for Embedded Systems.' In: *Instrumentation and Measurement, IEEE Transactions on* 62 (July 2013), pp. 1927–1934. DOI: 10.1109/TIM.2013.2248288.

[6] Mostafa Bazzaz, Mohammad Salehi, and Alireza Ejlali. 'An Accurate Instruction-Level Energy Estimation Model and Tool for Embedded Systems.' In: *Instrumentation and Measurement, IEEE Transactions on* 62 (July 2013), pp. 1927–1934. DOI: 10.1109/TIM.2013.2248288.

[7] L. Benini, A. Bogliolo, and G. De Micheli. 'System-level dynamic power management.' In: *Proceedings IEEE Alessandro Volta Memorial Workshop on Low-Power Design*. 1999, pp. 23–31. DOI: 10.1109/LPD.1999.750384.

[8] Yahia Benmoussa. 'Performance and Energy Consumption Characterization and Modeling of Video Decoding on Multi-core Heterogenous Mobile SoC and their Applications.' PhD thesis. June 2015. DOI: 10.13140/RG.2.1.4094.0640.

[9] Alberto Bosio, Arnaud Virazel, Patrick Girard, and Mario Barbareschi. 'Approximate computing: Design amp; test for integrated circuits.' In: *2017 18th IEEE Latin American Test Symposium (LATS)*. 2017, pp. 1–1. DOI: 10.1109/LATW.2017.7906737.

[10] D. Brooks, V. Tiwari, and Margaret Martonosi. 'Wattch: A framework for architectural-level power analysis and optimizations.' In: vol. 28. Feb. 2000, pp. 83–94. ISBN: 1-58113-232-8. DOI: 10.1109/ISCA.2000.854380.

[11] D. Burke, D. Jenkus, I. Qiqieh, R. Shafik, S. Das, and A. Yakovlev. 'Special session paper: significance-driven adaptive approximate computing for energy-e!icient image processing applications.' In: *2017 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. 2017, pp. 1–2. DOI: 10.1145/3125502.3125554.

[12] João M.P. Cardoso, José Gabriel F. Coutinho, and Pedro C. Diniz. 'Chapter 2 - High-performance embedded computing.' In: *Embedded Computing for High Performance*. Ed. by João M.P. Cardoso, José Gabriel F. Coutinho, and Pedro C. Diniz. Boston: Morgan Kaufmann, 2017, pp. 17–56. ISBN: 978-0-12-804189-5. DOI: https://doi.org/10.1016/B978-0-12-804189-5.00002-8. URL: https://www.sciencedirect.com/science/article/pii/B9780128041895000028.

[13] João M.P. Cardoso, José Gabriel F. Coutinho, and Pedro C. Diniz. 'Chapter 5 - Source code transformations and optimizations.' In: *Embedded Computing for High Performance*. Ed. by João Paiva Cardoso, José Figueired Coutinho, and Pedro C. Diniz. Boston: Morgan Kaufmann, 2017. ISBN: 978-0-12-804189-5. DOI: https://doi.org/10.1016/B978-0-12-804189-5.00002-8. URL: https://learning.oreilly.com/library/view/embedded-computing-for/9780128041994/B9780128041895000053.xhtml#s0015.

[14] Arun Chandrasekharan, Daniel Große, and Rolf Drechsler. 'ProACt: Hardware Architecture for Cross-Layer Approximate Computing.' In: *Design Automation Techniques for Approximation Circuits: Verification, Synthesis and Test*. Cham: Springer International Publishing, 2019, pp. 103–118. ISBN: 978-3-319-98965-5. DOI: 10.1007/978-3-319-98965-5_7. URL: https://doi.org/10.1007/978-3-319-98965-5_7.

[15] K. Chang and K. Wang. 'Enhancing performance of traffic safety guardian system on Android by task skipping mechanism.' In: *2013 IEEE International Symposium on Consumer Electronics (ISCE)*. 2013, pp. 115–116. DOI: 10.1109/ISCE.2013.6570137.

[16] Rituparna Chatterjee. 'How image processing will change your world in future.' In: *The Economic Times* (2011). URL: https://economictimes.indiatimes.com/tech/software/how-image-processing-will-change-your-world-in-future/articleshow/10394958.cms?from=mdr.

[17] Wei-Fan Chiang, Mark Baranowski, Ian Briggs, Alexey Solovyev, Ganesh Gopalakrishnan, and Zvonimir Rakamarić. 'Rigorous Floating-Point Mixed-Precision Tuning.' In: *SIGPLAN Not.* 52.1 (Jan. 2017), 300–315. ISSN: 0362-1340. DOI: 10.1145/3093333.3009846. URL: https://doi.org/10.1145/3093333.3009846.

[18] David Chinnery and Kurt Keutzer. 'Overview of the Factors Affecting the Power Consumption.' In: Jan. 2008, pp. 11–53. ISBN: 978-0-387-25763-1. DOI: 10.1007/978-0-387-68953-1_2.

[19] V. K. Chippa, D. Mohapatra, K. Roy, S. T. Chakradhar, and A. Raghunathan. 'Scalable Effort Hardware Design.' In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22.9 (2014), pp. 2004–2016. DOI: 10.1109/TVLSI.2013.2276759.

[20] Donald Craig. 1996. URL: http://www.cs.mun.ca/~donald/msc/node15.html.

[21] Oussama Djedidi, Mohand Djeziri, Nacer M'Sirdi, and Naamane Aziz. 'A Novel Easy-to-construct Power Model for Embedded and Mobile Systems - Using Recursive Neural Nets to Estimate Power Consumption of ARM-based Embedded Systems and Mobile Devices.' In: Jan. 2018, pp. 551–555. DOI: 10.5220/0006915805510555.

[22] Natalie Enright Jerger and Joshua San Miguel. 'Approximate Computing.' In: *IEEE Micro* 38.4 (2018), pp. 8–10. DOI: 10.1109/MM.2018.043191120.

[23] Mojtaba Farzmahdi and Rong Luo. 'Memoization-based high-performance video frame processing.' In: *Journal of Electronic Imaging* 25.6 (2016), pp. 1 –10. DOI: 10.1117/1.JEI.25.6.063025. URL: https://doi.org/10.1117/1.JEI.25.6.063025.

[24] Mojtaba Farzmahdi and Rong Luo. 'Memoization-based high-performance video frame processing.' In: *Journal of Electronic Imaging* 25.6 (2016), pp. 1 –10. DOI: 10.1117/1.JEI.25.6.063025. URL: https://doi.org/10.1117/1.JEI.25.6.063025.

[25] I. Felzmann, J. F. Filho, and L. Wanner. 'Risk-5: Controlled Approximations for RISC-V.' In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.11 (2020), pp. 4052–4063. DOI: 10.1109/TCAD.2020.3012312.

[26] *FireSim Simulator*. https://github.com/firesim/firesim. Accessed: 2021-02-01.

[27] A. Ghosh, M. Bershteyn, R. Casley, C. Chien, A. Jain, M. Lipsie, D. Tarrodaychik, and O. Yamamoto. 'A Hardware-Software Co-simulator for Embedded System Design and Debugging.' In: *Readings in Hardware/Software Co-Design*. Ed. by Giovanni De Micheli, Rolf Ernst, and Wayne Wolf. Systems on Silicon. San Francisco: Morgan Kaufmann, 2002, pp. 569–578. DOI: https://doi.org/10.1016/B978-155860702-6/50052-1. URL: https://www.sciencedirect.com/science/article/pii/B9781558607026500521.

[28] Ed Grochowski and Murali Annavaram. 'Energy per instruction trends in Intel microprocessors.' In: *Technology@ Intel Magazine* 4.3 (2006), pp. 1–8.

[29] R.X. Gu and M.I. Elmasry. 'Power dissipation analysis and optimization of deep submicron CMOS digital circuits.' In: *IEEE Journal of Solid-State Circuits* 31.5 (1996), pp. 707–713. DOI: 10.1109/4.509853.

[30] Amira Guesmi, Ihsen Alouani, Khaled N. Khasawneh, Mouna Baklouti, Tarek Frikha, Mohamed Abid, and Nael Abu-Ghazaleh. 'Defensive approximation: securing CNNs using approximate computing.' In: *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (2021). DOI: 10.1145/3445814.3446747. URL: http://dx.doi.org/10.1145/3445814.3446747.

[31] Chen Guo, Song Ci, Yanglin Zhou, and Yang Yang. 'A Survey of Energy Consumption Measurement in Embedded Systems.' In: *IEEE Access* 9 (2021), pp. 60516–60530. DOI: 10.1109/ACCESS.2021.3074070.

[32] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy. 'IMPACT: IMPrecise adders for low-power approximate computing.' In: *IEEE/ACM International Symposium on Low Power Electronics and Design*. 2011, pp. 409–414. DOI: 10.1109/ISLPED.2011.5993675.

[33] Jie Han and Michael Orshansky. 'Approximate computing: An emerging paradigm for energy-efficient design.' In: *2013 18th IEEE European Test Symposium (ETS)*. 2013, pp. 1–6. DOI: 10.1109/ETS.2013.6569370.

[34] Bingsheng He. 'When Data Management Systems Meet Approximate Hardware: Challenges and Opportunities.' In: *Proc. VLDB Endow.* 7.10 (June 2014), 877–880. ISSN: 2150-8097. DOI: 10.14778/2732951.2732961. URL: https://doi-org.proxy.library.carleton.ca/10.14778/2732951.2732961.

[35] Xin He, Syed Kadry, and Afshin Abdollahi. 'Adaptive leakage control on body biasing for reducing power consumption in CMOS VLSI circuit.' In: Mar. 2009, pp. 465–470. DOI: 10.1109/ISQED.2009.4810339.

[36] N. Ho, E. Manogaran, W. Wong, and A. Anoosheh. 'Efficient floating point precision tuning for approximate computing.' In: *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2017, pp. 63–68. DOI: 10.1109/ASPDAC.2017.7858297.

[37] J. Huang, T. N. Kumar, and H. Abbas. 'A promising power-saving technique: Approximate computing.' In: *2018 IEEE Symposium on Computer Applications Industrial Electronics (ISCAIE)*. 2018, pp. 285–290. DOI: 10.1109/ISCAIE.2018.8405486.

[38] J. Huang, T. Nandha Kumar, and H. Abbas. 'Simulation-Based Evaluation of Approximate Adders for Image Processing Using Voltage Overscaling Method.' In: *2020 IEEE 5th International Conference on Signal and Image Processing (ICSIP)*. 2020, pp. 499–505. DOI: 10.1109/ICSIP49896.2020.9339354.

[39] Shariq Hussain. 'Energy Optimization for Low Power Embedded Systems.' In: *Journal of Information Engineering and Applications* (2015), 89–92. URL: https://core.ac.uk/reader/234677245.

[40] Oliver C. Ibe. 'Chapter 12 - Special Random Processes.' In: *Fundamentals of Applied Probability and Random Processes (Second Edition)*. Ed. by Oliver C. Ibe. Second Edition. Boston: Academic Press, 2014, pp. 369–425. ISBN: 978-0-12-800852-2. DOI: https://doi.org/10.1016/B978-0-12-800852-2.00012-2. URL: https://www.sciencedirect.com/science/article/pii/B9780128008522000122.

[41] Muhammed Ibrahim and Ibrahim Hamarash. 'Dynamic voltage frequency scaling (DVFS) for microprocessors power and energy reduction.' In: (Dec. 2005).

[42] Hailin Jiang, M. Marek-Sadowska, and S.R. Nassif. 'Benefits and costs of power-gating technique.' In: *2005 International Conference on Computer Design*. 2005, pp. 559–566. DOI: 10.1109/ICCD.2005.34.

[43] Mahmut Kandemir, N. Vijaykrishnan, and Mary Jane Irwin. 'Compiler Optimizations for Low Power Systems.' In: *Power Aware Computing*. USA: Kluwer Academic Publishers, 2002, 191–210. ISBN: 0306467860.

[44] F. Khalvati, Aagaard, M.D., and Tizhoosh. 'Window memoization: toward high-performance image processing software. J Real-Time Image Proc.' In: 2015. DOI: https://doi.org/10.1007/s11554-012-0247-8. URL: https://link.springer.com/article/10.1007/s11554-012-0247-8.

[45] T. Kitahara, F. Minami, T. Ueda, K. Usami, S. Nishio, M. Murakata, and T. Mitsuhashi. 'A clock-gating method for low-power LSI design.' In: *Proceedings of 1998 Asia and South Pacific Design Automation Conference*. 1998, pp. 307–312. DOI: 10.1109/ASPDAC.1998.669476.

[46] Ulrich Kremer. 'Low- Power/ Energy Compiler Optimizations.' In: Nov. 2004, pp. 35–1. ISBN: 978-0-8493-1941-9. DOI: 10.1201/9781420039559.ch35.

[47] Saurabh Kshirsagar and M.B. Dr. Mali. 'A Review of Clock Gating Techniques in Low Power Applications.' In: *International Journal of Innovative Research in Science,Engineering and Technology* 4.6 (2015), pp. 1–5. DOI: 10.15680/IJIRSET.2015.0406103.

[48] Roger Vicente Caputo LLANOS. 'Voltage Scaling Interfaces for Multi-Voltage Digital Systems.' PhD thesis. 2015.

[49] Etienne Le Sueur and Gernot Heiser. 'Dynamic Voltage and Frequency Scaling: The Laws of Diminishing Returns.' In: *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*. HotPower'10. Vancouver, BC, Canada: USENIX Association, 2010, 1–8.

[50] Sheayun Lee, Andreas Ermedahl, Sang Lyul Min, and Naehyuck Chang. 'An Accurate Instruction-Level Energy Consumption Model for Embedded RISC Processors.' In: *Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems*. LCTES '01. Snow Bird, Utah, USA: Association for Computing Machinery, 2001, 1–10. ISBN: 1581134258. DOI: 10.1145/384197.384201. URL: https://doi.org/10.1145/384197.384201.

[51] Li Li and Hai Zhou. 'On error modeling and analysis of approximate adders.' In: *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2014, pp. 511–518. DOI: 10.1109/ICCAD.2014.7001399.

[52] R. Li, R. Naous, H. Fariborzi, and K. N. Salama. 'Approximate Computing With Stochastic Transistors' Voltage Over-Scaling.' In: *IEEE Access* 7 (2019), pp. 6373–6385. DOI: 10.1109/ACCESS.2018.2889747.

[53] Shikai Li, Sunghyun Park, and Scott Mahlke. 'Sculptor: Flexible Approximation with Selective Dynamic Loop Perforation.' In: *Proceedings of the 2018 International Conference on Supercomputing*. ICS '18. Beijing, China: Association for Computing Machinery, 2018, 341–351. ISBN: 9781450357838. DOI: 10.1145/3205289.3205317. URL: https://doi-org.proxy.library.carleton.ca/10.1145/3205289.3205317.

[54] David Lidsky and Jan M. Rabaey. 'Early Power Exploration—a World Wide Web Application.' In: *Proceedings of the 33rd Annual Design Automation Conference*. DAC '96. Las Vegas, Nevada, USA: Association for Computing Machinery, 1996, 27–32. ISBN: 0897917790. DOI: 10.1145/240518.240523. URL: https://doi-org.proxy.library.carleton.ca/10.1145/240518.240523.

[55] Liming Lou, Paul Nguyen, Jason Lawrence, and Connelly Barnes. 'Image Perforation: Automatically Accelerating Image Pipelines by Intelligently Skipping Samples.' In: vol. 35. 5. New York, NY, USA: Association for Computing Machinery, Sept. 2016. DOI: 10.1145/2904903. URL: https://doi-org.proxy.library.carleton.ca/10.1145/2904903.

[56] T. Moreau, J. San Miguel, M. Wyse, J. Bornholt, A. Alaghi, L. Ceze, N. Enright Jerger, and A. Sampson. 'A Taxonomy of General Purpose Approximate Computing Techniques.' In: *IEEE Embedded Systems Letters* 10.1 (2018), pp. 2–5. DOI: 10.1109/LES.2017.2758679.

[57] Thierry Moreau, Mark Wyse, Jacob Nelson, Adrian Sampson, Hadi Esmaeilzadeh, Luis Ceze, and Mark Oskin. 'SNNAP: Approximate computing on programmable SoCs via neural acceleration.' In: *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. 2015, pp. 603–614. DOI: 10.1109/HPCA.2015.7056066.

[58] P. Nad', A. Dhananjaya, and S. SumaM. 'Optimization of Delay and Leakage using Body Bias.' In: *International journal of engineering research and technology* 2 (2013).

[59] Geneviève Ndour, Tiago Trevisan Jost, Anca Molnos, Yves Durand, and Arnaud Tisserand. 'Evaluation of variable bit-width units in a RISC-V processor for approximate computing.' In: Apr. 2019, pp. 344–349. DOI: 10.1145/3310273.3323159.

[60] Aleksandr Ometov et al. 'A Survey on Wearable Technology: History, State-of-the-Art and Current Challenges.' In: *Computer Networks* 193 (2021), p. 108074. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2021.108074. URL: https://www.sciencedirect.com/science/article/pii/S1389128621001651.

[61] Reza Omidi and Sepehr Sharifzadeh. 'Design of low power approximate floating-point adders.' In: *International Journal of Circuit Theory and Applications* 49.1 (2021), pp. 185–195. DOI: https://doi.org/10.1002/cta.2831. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cta.2831. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cta.2831.

[62] Robert Oshana. 'Optimizing Embedded Software for Power.' In: *Software Engineering for Embedded Systems*. Elsevier.

[63] Jongse Park, Hadi Esmaeilzadeh, Xin Zhang, Mayur Naik, and William Harris. 'FlexJava: Language Support for Safe and Modular Approximate Programming.' In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE 2015. Bergamo, Italy: Association for Computing Machinery, 2015, 745–757. ISBN: 9781450336758. DOI: 10.1145/2786805.2786807. URL: https://doi.org/10.1145/2786805.2786807.

[64] Abhinav Pathak, Y. Charlie Hu, Ming Zhang, Paramvir Bahl, and Yi-Min Wang. 'Fine-Grained Power Modeling for Smartphones Using System Call Tracing.' In: *Proceedings of the Sixth Conference on Computer Systems*. EuroSys '11. Salzburg, Austria: Association for Computing Machinery, 2011, 153–168. ISBN: 9781450306348. DOI: 10.1145/1966445.1966460. URL: https://doi-org.proxy.library.carleton.ca/10.1145/1966445.1966460.

[65] Massoud Pedram. 'Power Optimization and Management in Embedded Systems.' In: *Proceedings of the 2001 Asia and South Pacific Design Automation Conference*. ASP-DAC '01. Yokohama, Japan: Association for Computing Machinery, 2001, 239–244. ISBN: 0780366344. DOI: 10.1145/370155.370333. URL: https://doi-org.proxy.library.carleton.ca/10.1145/370155.370333.

[66] 'Power Gating Overview.' In: *Low Power Methodology Manual: For System-on-Chip Design*. Boston, MA: Springer US, 2007, pp. 33–40. ISBN: 978-0-387-71819-4. DOI: 10.1007/978-0-387-71819-4_4. URL: https://doi.org/10.1007/978-0-387-71819-4_4.

[67] *QEMU Simulator*. https://gitlab.com/qemu-project/qemu. Accessed: 2021-02-01.

[68] Qinru Qiu, Qing Wu, and M. Pedram. 'Dynamic power management of complex systems using generalized stochastic petri nets.' In: *Proceedings 37th Design Automation Conference*. 2000, pp. 352–356. DOI: 10.1109/DAC.2000.855335.

[69] 'RISC-V OFFERS SIMPLE, MODULAR ISA.' In: 2016.

[70] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. 'EnerJ: Approximate Data Types for Safe and General Low-Power Computation.' In: *SIGPLAN Not.* 46.6 (June 2011), 164–174. ISSN: 0362-1340. DOI: 10.1145/1993316.1993518. URL: https://doi-org.proxy.library.carleton.ca/10.1145/1993316.1993518.

[71] Adrian Sampson, André Baixo, Benjamin Ransford, Thierry Moreau, Joshua Yip, Luis Ceze, and Mark Oskin. 'ACCEPT: A programmer-guided compiler framework for practical approximate computing.' In: *University of Washington Technical Report UW-CSE-15-01* 1 (2015).

[72] Muhammad Shafique, Rehan Hafiz, Semeen Rehman, Walaa El-Harouni, and Jörg Henkel. 'Invited: Cross-layer approximate computing: From logic to architectures.' In: *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. 2016, pp. 1–6. DOI: 10.1145/2897937.2905008.

[73] Zayan Shaikh and Ehsan Atoofian. 'Approximate trivial instructions.' In: *Proceedings of the 17th ACM International Conference on Computing Frontiers*. 2020, pp. 1–9.

[74] Joseph J. Sharkey, Dmitry V. Ponomarev, Kanad Ghose, and Oguz Ergin. 'Instruction Packing: Toward Fast and Energy-Efficient Instruction Scheduling.' In: *ACM Trans. Archit. Code Optim.* 3.2 (June 2006), 156–181. ISSN: 1544-3566. DOI: 10.1145/1138035.1138037. URL: https://doi.org/10.1145/1138035.1138037.

[75] Stelios Sidiroglou-Douskos, Sasa Misailovic, Henry Hoffmann, and Martin Rinard. 'Managing Performance vs. Accuracy Trade-Offs with Loop Perforation.' In: *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*. ESEC/FSE '11. Szeged, Hungary: Association for Computing Machinery, 2011, 124–134. ISBN: 9781450304436. DOI: 10.1145/2025113.2025133. URL: https://doi-org.proxy.library.carleton.ca/10.1145/2025113.2025133.

[76] Kevin Skadron, Margaret Martonosi, David I. August, Mark D. Hill, David J. Lilja, and Vijay S. Pai. 'Challenges in computer architecture evaluation.' English (US). In: *Computer* 36.8 (Aug. 2003), pp. 30–36. ISSN: 0018-9162. DOI: 10.1109/MC.2003.1220579.

[77] Wlad Sobol. 'Recent advances in MRI technology: Implications for image quality and patient safety.' In: *Saudi journal of ophthalmology : official journal of the Saudi Ophthalmological Society* 26 (Oct. 2012), pp. 393–9. DOI: 10.1016/j.sjopt.2012.07.005.

[78] *Spike Simulator*. https://github.com/riscv/riscv-isa-sim. Accessed: 2021-02-01.

[79] Embedded Staff. *Optimizing embedded software for power efficiency: Part 1 – measuring power*. 2014. URL: https://www.embedded.com/optimizing-embedded-software-for-power-efficiency-part-1-measuring-power/.

[80] *THE USC-SIPI Image Database*. http://sipi.usc.edu/database/. Accessed: 2021-06-01.

[81] Mike Thomas. *A Deep Dive Into the World of Computer Simulations*. URL: https://builtin.com/hardware/computer-simulation.

[82] Marcello Traiola, Alessandro Savino, and Stefano Di Carlo. 'Probabilistic estimation of the application-level impact of precision scaling in approximate computing applications.' In: *Microelectronics Reliability* 102 (2019), p. 113309. ISSN: 0026-2714. DOI: https://doi.org/10.1016/j.microrel.2019.06.002. URL: https://www.sciencedirect.com/science/article/pii/S0026271418309442.

[83] *U.S. Energy Information Administration - EIA - Independent Statistics and Analysis*. URL: https://www.eia.gov/todayinenergy/detail.php?id=42342.

[84] Evangelos Vasilakis. *An Instruction Level Energy Characterization of ARM Processors*. 2015.

[85] Minh Huan Vo. 'Comparative Study on Power Gating Techniques for Lower Power Delay Product, Smaller Power Loss, Faster Wakeup Time.' In: *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems* 5.15 (Aug. 2018). DOI: 10.4108/eai.27-6-2018.155236.

[86] Andrew Waterman, Yunsup Lee, David A. Patterson, and Krste Asanovi. 'The RISC-V Instruction Set Manual. Volume 1: User-Level ISA, Version 2.0.' In: 2014.

[87] Li-Chuan Weng, XiaoJun Wang, and Bin Liu. 'A survey of dynamic power optimization techniques.' In: *The 3rd IEEE International Workshop on System-on-Chip for Real-Time Applications, 2003. Proceedings.* 2003, pp. 48–52. DOI: 10.1109/IWSOC.2003.1213004.

[88] Wikipedia. *Processor power dissipation*. [Online; accessed 22-July-2021]. 2004. URL: https://en.wikipedia.org/wiki/Processor_power_dissipation.

[89] Mark Wyse. 'Modeling Approximate Computing Techniques.' In: 2015.

[90] Amir Yazdanbakhsh et al. 'Axilog: language support for approximate hardware design.' In: *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. 2015, pp. 812–817.

[91] Amir Yazdanbakhsh, Divya Mahajan, Hadi Esmaeilzadeh, and Pejman Lotfi-Kamran. 'AxBench: A Multiplatform Benchmark Suite for Approximate Computing.' In: *IEEE Design Test* 34.2 (2017), pp. 60–68. DOI: 10.1109/MDAT.2016.2630270.

[92] Ning Zhu, Wang Ling Goh, and Kiat Seng Yeo. 'An enhanced low-power high-speed Adder For Error-Tolerant application.' In: *Proceedings of the 2009 12th International Symposium on Integrated Circuits*. 2009, pp. 69–72.

[93] *gem5 Simulator*. https://gem5.googlesource.com/public/gem5/. Accessed: 2021-02-01.

[94] *i.MX 6Solo/6DualLite Applications Processors for Industrial Products*. IMX6SDLIEC. Rev. 9. NXP Semiconductors. 2018. URL: https://www.nxp.com/docs/en/data-sheet/IMX6SDLIEC.pdf.

[95] *riscvOVPsim Simulator*. https://github.com/riscv-ovpsim. Accessed: 2021-02-01.