BROWN UNIVERSITY

Exploiting Timing Violations under Voltage Scaling with Hardware Transactional Memory

by

Sungseob Whang

A thesis submitted in partial fulfillment for the
Degree of Bachelor of Science with Honors in Computer Science

Advisor: R. Iris Bahar
School of Engineering

December 2017

BROWN UNIVERSITY

Abstract

Advisor: R. Iris Bahar
School of Engineering

Exploiting Timing Violations under Voltage Scaling with Hardware Transactional Memory

by Sungseob Whang

Energy consumption is the dominant factor in many computing systems. Voltage scaling is a widely used technique to lower power dissipation, which exploits supply voltage margins that ensure reliable circuit execution. Aggressive voltage scaling will slow signal propagation; without coherent frequency relaxation, timing violations may be generated. *Hardware Transactional Memory* (HTM) offers an error recovery mechanism that allows reliable execution and power savings with modest overhead. We examine how timing violations may be generated in specific hardware environments such as floating point adders and multipliers and develop *Critical Bit* error model to accurately model erroneous behavior when timing violations are permitted. Simulating from such error model, we propose a hardware/software environment *IgnoreTM*, where timing violations are opportunistically ignored, allowing for more aggressive voltage scaling. Experiments show our technique allows 32% to 45% power savings compared to non-voltage-scaled simulations, and 5% to 15% additional power savings compared to existing HTM solutions without impacting runtime and fine-tuned accuracy loss.

Acknowledgements

This work is supported in part by NSF under Grants CSR-1319095 and CSR-1519576.

I personally thank my advisor Professor R. Iris Bahar, Professor Tali Moreshet (Boston University), and Professor Dimitra Papagiannopoulou (University of Massachusetts, Lowell) for their extraordinary guidance and effort that were crucial to the culmination of this thesis.

I also thank Professor Maurice Herlihy, Jiwon Choe at Brown University, Dr. Andrea Marongiu at ETH Zurich, and other PhD candidates at Brown University who provided expertise that greatly assisted the research.

Contents

List of Figures

Chapter 1

Introduction

Energy consumption is a dominant constraint of any computing system, from improving the battery life of embedded systems to reducing the power demands of servers. Transistor sizes are minimized in order to maximize performance of such systems, that lead to devices being more susceptible to the effects of static and dynamic variability [1]. To ensure reliable operation regarding these variability, designers have conservatively added safety margins, or guardbands, of clock frequency, supply voltage and temperature that constrain the system to operate in non-optimal fashion. The choice of guardband design accommodated with simple error correction/detection mechanisms (such as in SRAMs) provided robustness and reliability of the system at the cost of reduced yield, performance and energy efficiency. Previous works have proposed various ways of timing error detection and correction techniques at hardware and software levels in order to obtain energy efficiency with voltage scaling. Latest work from our research group [13] developed methodologies of utilizing Hardware Transactional Memory(HTM) - originally designed for memory synchronization - as an error recovery mechanism to easily control supply voltage levels and ensure correct computation.

However, these previous work do not investigate energy reductions from exploiting the inexact-ness of computation. *Approximate computing* is a newly researched area based on the observation that exact computation and perfect accuracy are not always necessary. Inherent tolerance to inaccurate computation exists for specific applications in the areas including but not limited to: machine learning, signal processing, image processing, scientific computing, data mining, and data analysis. These applications can be examined further than exact applications by trading accuracy with both performance and energy consumption. However, the approximation in hardware level poses some key challenges as it requires understanding of which specific instructions and data can be approximated, and not introducing severe control flow glitches that can possibly crash the program and even, the system.

This thesis investigates the biggest challenges of exploiting timing violations and allowing approximation using HTM and voltage scaling in order to save energy. By configuring a target platform and performing timing error analysis, we introduce *IgnoreTM*, a hardware-software co-design framework that connects approximation with timing-induced errors, using the lightweight error recovery mechanism from HTM to reach maximum energy efficiency.

This thesis makes the following contributions:

- We analyze high-level implementations of floating point adder and multiplier aiming to realistically simulate bit-level timing violations from voltage scaling, and introduce the *Critical Bit* error model.
- We develop a simple interface for the programmer to clarify what level of approximation is wanted, to the level of *how many timing errors per specific code snippet*, with flexibility to differ approximation thresholds throughout different code portions while being able to integrate HTM for concurrency.
- We create a hardware-software environment that supports the above interface, and develop an algorithm that dynamically optimizes energy consumption from a combination of static and dynamic monitors to determine appropriate conditions for voltage adjustments such that approximate result is within tolerable bounds of the exact result.
- We concentrate on floating-point applications, which have potential of inherent tolerance to inaccuracy, and demonstrate these applications can benefit from voltage scaling with controlled accuracy loss.
- We show that an additional 4-15% energy savings is possible using our approach, compared to always correcting errors, with a very modest 1% performance overhead compared to no voltage scaling.

The organization of the remainder of this thesis is as follows: Chapter 2 explains the background and previous work done in approximation and Transactional Memory; Chapter 3 explains the target architecture and implementation of system necessary to utilize our technique; Chapter 4 characterize timing violation behavior of floating point adder and multiplier and compare it to previously used timing error models; Chapter 5 explains how approximation and energy consumption is controlled in the target platform, comparing them to previous voltage scaling policies; Chapter 6 presents simulation results of our proposed design and discusses the benefits of our approach compared to previous solutions; Chapter 7 concludes this thesis and introduce interesting problems for future work.

Chapter 2

Background

Approximate computing is based on the observation that for many applications, deliberately allowing some level of computational inaccuracy and providing an approximate result is acceptable, and can improve performance and energy consumption. Many application domains, including video, signal and image processing, machine learning, computer vision, data mining and analysis, and gaming, are inherently tolerant to inaccuracy and have an intrinsic resilience to errors. Studies have shown that many of these applications have portions which are more resilient to hardware and software errors, and other portions that are critical and must be protected from errors. With approximate programming we can exploit the relaxed precision requirements of such applications in order to increase performance and reduce energy. Various approximate computing techniques have been proposed in literature, on different levels of the system design. Software approaches propose approximate programming models combined with programming level techniques for approximating code regions, such as loop perforation, memoization, task skipping, data packing, relax synchronization, etc. They suggest real benefits from approximate computation, and are focused on modifying the code itself to compute approximate versions of the output. At the hardware level, there have been proposals for inexact circuits and faulty hardware, such as adders, multipliers, dividers, and other components [14, 12, 11, 10, 9, 16]. These approaches differ from our work that they all require either some algorithmic changes to the software, or integration of complex dedicated approximate hardware. Instead, we propose minimal change to the software such that the burden for the programmer is low, while utilizing minimal hardware transactional memory circuitry and voltage rails to enable pure voltage-scaling-induced approximation, and focusing on energy reduction as our goal.

Another aspect that differentiates our project from existing works is the accuracy of the error models used. Many of these works follow simplistic error models such as single bit-flip probabilities, uniform distribution models or random values [4, 16, 17] that are not able to fully capture the error behavior of functional units (FUs). For example, the error models considered in EnerJ [16] are basic fault injection error models with single bit flips at the output using uniform distribution, random selection or previously seen values for the output. The work in [4] relies on uniform bit-error models with set error probabilities per component. In [6] the authors develop a model for the expected probability of errors due to timing violations when the supply voltage is reduced in integer adders and multipliers, but this model does not take into account bit location and history of computation. Most of the existing error models do not cover important families of FUs, such as Floating Point Units (FPUs) and bit-wise logic operation units, which have different behavior than integer adders. Moreover, these models ignore three important factors: value correlation, computation history, and bit-wise error variability [18]. Tziantzioulis et al. [18] introduced b-HiVE, an error model that considers these three factors and showed that the bit-wise error rate of a functional unit's operation can be predicted more accurately. However, b-Hive relies on the independence of bit-wise error rates. In our recent work [19] which is summarized in Chapter 4, we address these limitations and introduce a new error model that entails these concerns by analyzing specific hardware and critical timing paths. Constructing these error models is paramount in understanding and characterizing the error tolerance of applications under different approximations.

Chapter 3

Target Architecture and High-Level Implementation

3.1 PULP Architecture

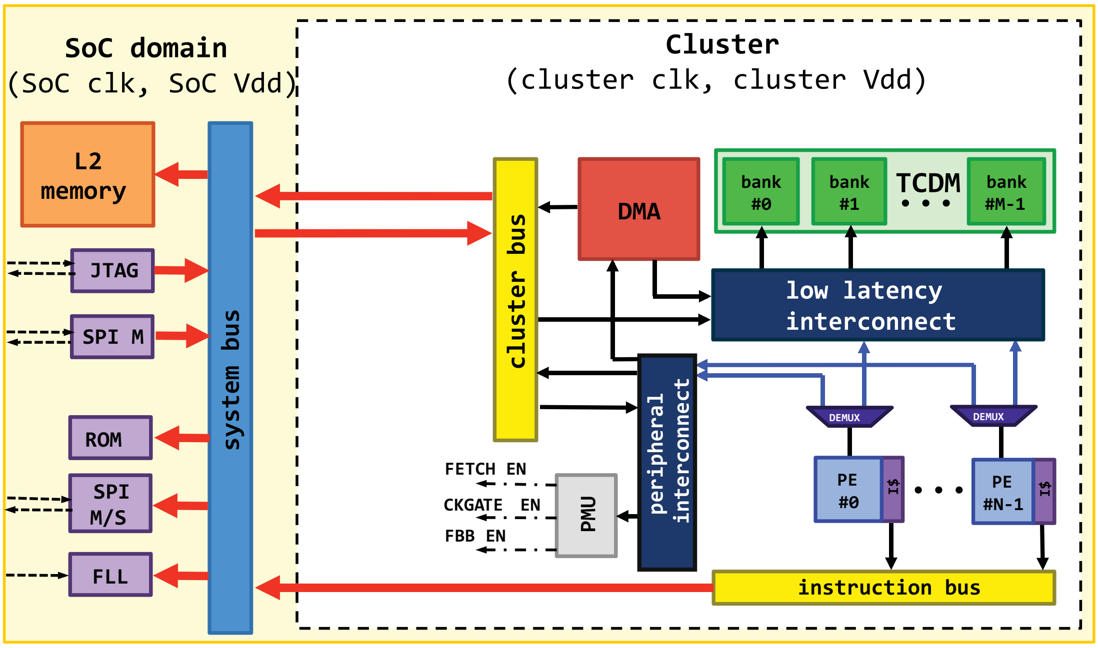

FIGURE 3.1: PULP System-on-Chip architecture.

While we believe that our approach is applicable to a range of platforms, we focus here on a cluster-based many-core system appropriate for embedded platforms. We chose this particular testbed both because it is highly parallel, and because embedded systems tend to be energy-constrained. The target architecture is based on *PULP* (Parallel Ultra Low Power platform), a scalable computing platform designed to operate on a large range of operating voltages [15]. PULP leverages tightly-coupled shared-memory *clusters* as a main building block. The latest physical implementation of the platform, the PULP3 SoC, is shown in Figure 3.1. PULP3 features a single computing *cluster* integrated with 64 kB of L2 SRAM memory and several I/O peripherals (e.g., DMA) accessible through a system bus. The core is based on a power-optimized implementation of the OpenRISC-ISA; multiple cores share a single instruction cache (*I$*). To avoid the energy overhead of memory coherency, the cores do not have private data caches; they all share a multi-banked tightly coupled data memory (*TCDM*) configured as a shared data scratchpad. Intra-cluster communication is based on a high bandwidth *low-latency interconnect*, implementing a word-level interleaving scheme to reduce access contention to the TCDM.

3.2 HTM Infrastructure

An underlying runtime system (RTS) transparently manages the transactions with a core-level policy that optimistically lowers the voltage in small steps. A snapshot of the system state is taken before each transaction is started so that if the allowed error threshold is exceeded during transaction execution, this safe state can be restored. For error detection, we assume that each core is equipped with runtime error-detection circuitry, such as error-detection sequential (EDS) [2]. We handle these errors at the granularity of a transaction. If the acceptable error threshold is exceeded while executing a transaction, the system aborts that transaction, takes the required countermeasures in terms of voltage settings, and restarts the transaction.

Traditionally, HTM design is based on three key components: (1) a bookkeeping mechanism to keep track of read/write data accesses and detect conflicts, (2) a data versioning technique to keep track of original and speculative data versions for recovery in case of conflicts and (3) a check-pointing and rollback mechanism to recover from data conflicts and retry failed transactions. Since we are not using HTM for memory synchronization, for this preliminary study we bypass the bookkeeping mechanism, obtaining a more lightweight design.

Checkpointing and Rollback We enclose each block of the program that has instructions that can be approximated, within a transaction. At the beginning of each transaction, we save the core's internal state (program counter, stack pointer, registers, stack contents) to be able to roll back in case the allowed error threshold is exceeded. If the error threshold is not exceeded, the transaction *commits*, the checkpoint is discarded, and speculative changes to the data become permanent. If the error threshold is exceeded, the transaction *aborts*, and a *rollback* mechanism restores the internal core state. In addition, data are restored to their original values and speculative copies are discarded.

Data Versioning We use a distributed logging scheme to enable data versioning [13]. Logs are distributed among the TCDM banks and each bank keeps a fixed-size log space for each core in the system. Note that only the first time an address is written does its original value need to be saved in the log, so the log size is quite modest. In practice, the logs have been shown to occupy only about 3% of the total TCDM space. The log saving and restoration process is done independently at each memory bank and does not require interaction with other banks, which makes it very fast and efficient.

3.3 Floating Point Unit Integration

Each processor is coupled with a floating point unit following the specifications of VFP coprocessor instruction set of ARMv3 and higher. Each coprocessor is modeled with realistic delay for each instruction following the ARM7100 floating point coprocessor specification, as floating point instructions generally need more propagation delay as the circuitry is more complex. For example, multiply-accumulate is modeled as 8 cycles, multiply is modeled as 5 cycles, and add is modeled as 4 cycles. These cycles are not pipelined, and each instruction that takes more than 1 cycle would block the processor instruction stream until it is finished, which models the behavior of a fully combinatoric design of a simple ARM processor.

We use the floating point unit as the core of approximation, of non-critical errors. Chapter 4 analyzes floating point timing violations and Chapter 5 introduces critical and non critical timing violations - as differentiation of critical and non-critical errors in a normal integer instruction stream is complex, we standardize the differentiation through the module-level such that any floating point instruction can be approximated, while other instructions we assume that they are critical. Hence, to fully exploit the approximation boundaries of the floating point unit, we employ separate voltage rails for the floating point unit and the processor, such that the processor can stay in a higher supply voltage (which is scaled) as they execute critical instructions, while the floating point unit supply voltage can be more aggressively scaled as they execute non-critical instructions.

Chapter 4

Timing Error Model of Floating Point Units

We consider a variety of error models from earlier work [16, 17, 7, 18], as well as our *Critical Bit* model, to generate erroneous behavior of floating point arithmetic units. This chapter summarizes a significant portion of [19].

4.1 Previuos Error Models

We examine the following previously used error models:

- Random Model: EnerJ [16] introduced 3 widely used error injection models: *Random*, *Previous* and *Single* [17, 18]. Each model assumes an error rate p, and when the error occurs, the result is switched to a random value, the previous operations' result, or a randomly selected single-bit-flipped value, respectively. For our testing purposes, we consider the random value model to compare with other error models.

- Uniform Model: b-HiVE's interpretation of Lane-Decouple's model [7, 18] used a uniform error model, where each bit of the result is independently flipped for a bit-wise uniform error rate p. Rahimi [14] added a *bit-boundary*, assuming only bits between a range from bit location a to bit location b have probability of flipping with a bit-wise uniform error rate, while the remaining bit locations do not flip. The *bit-boundary* is implemented in OpenMP on an FPU architecture by removing the fault detection circuitry of certain result bit locations.

- b-HiVE Model: b-HiVE (bit-level History-based Error model with Value Correlation) runs a trace on a functional unit under a low-voltage environment and for each bit, observes the previous computational result, the previous latched result, the current computational result and the current latched result, and classifies the 4-tuple into 5 categories: *Correct, Previous Observed, Previous Correct, Glitch*, and *Ambiguous*. Using this model, [18] and [8] compute the bit-wise flip rates of integer addition and floating point addition and multiplication. We use these results to model the behavior of floating point multiplication and addition by specifying bit-wise error rates for each voltage level, without taking history into account.

4.2 Critical Bit Model

In this paper, we also introduce a new bit-wise dependent error model of floating point operations, called *'Critical bit flip'* model, where the flip rate at each bit location is dependent on the current and the previous computation of the *critical bit*. We assume that, based on the architecture of floating point arithmetic units, bit flips should be dependent on some intermediate bit/wire that is on the critical path of the computation. The *critical bits* of an arithmetic unit are responsible for the simultaneous bit flips in the result when the voltage is gradually scaled down. Next, we explain in detail how we choose these critical bits in different arithmetic units and how we construct an error model based on that choice. We construct our error models using the following process:

1. We assume a specific architecture for each arithmetic operation,
2. We determine the critical paths of the architecture,
3. We determine sensitive critical paths with similar delays, and the intermediate bits (*i.e., 'critical bits'*) responsible for those paths,
4. We assume that the critical bit is the only possible destination of the timing error, thus if unchanged from the previous operation, it would latch the correct result,
5. We assume that the critical bit changed from the previous operation would latch the result computed with the critical bit flipped.

In order to design error models for each arithmetic operation, we next define the critical bit paths for each arithmetic unit, i.e. for floating-point addition and multiplication.

4.2.1 Critical Bits of Floating Point Multiplier

A 32-bit floating point number is represented with 3 parts: a 1-bit sign (1 for negative, 0 for positive), an 8-bit exponent (00000000 for –127, 11111111 for +128, biased by –127), and a 23-bit mantissa. A 32-bit floating point multiplier appends a 1 to the 23-bit mantissa and multiplies the two 24-bit mantissas treating them as fixed point integers. The result becomes the mantissa of the output. The exponents are separately added to determine the result's exponent (Fig 4.1).

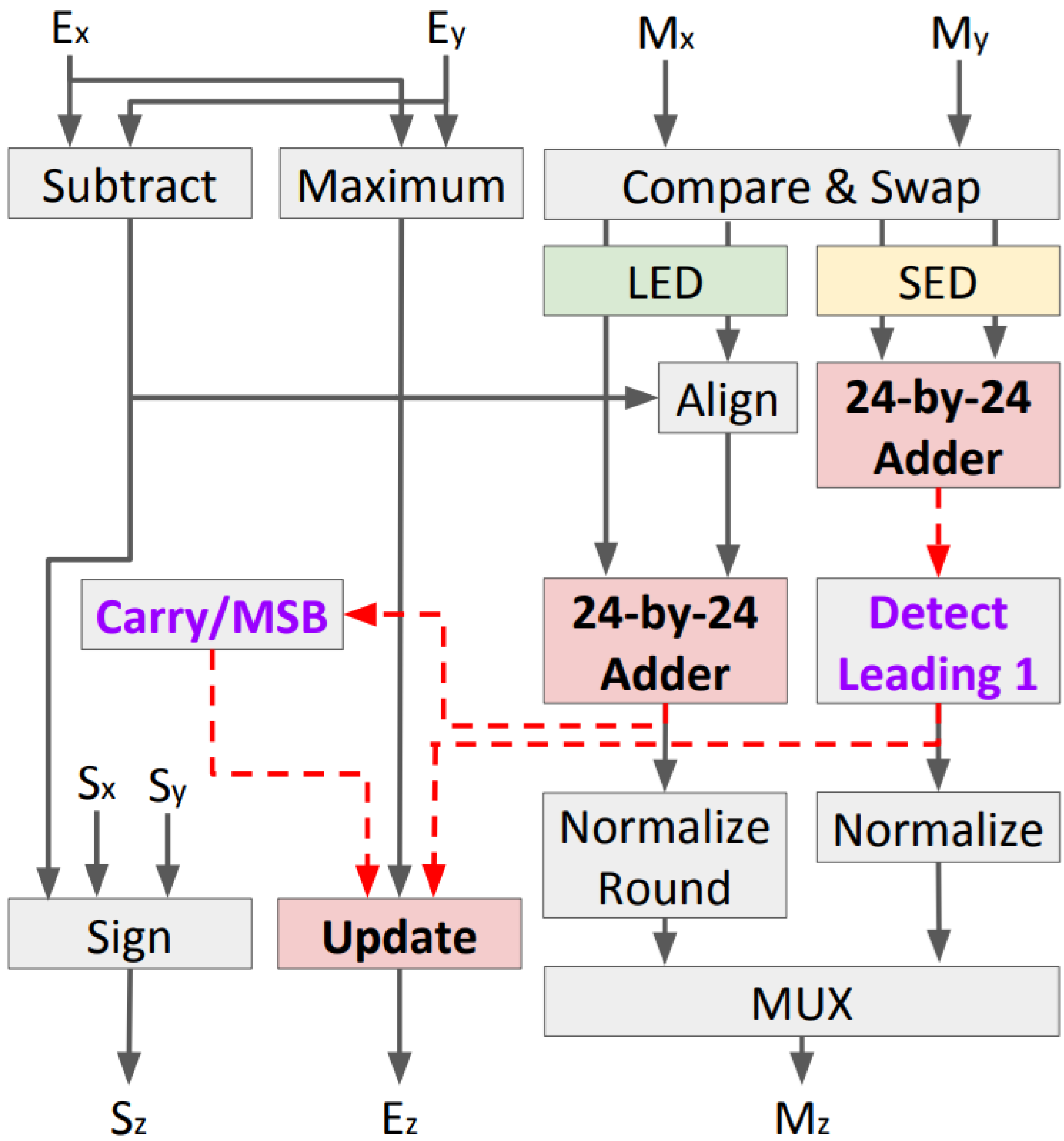

FIGURE 4.1: Floating-point multiplication architecture

Each operand of the 24-bit fixed point multiplier belongs in the range $[2^{23}, 2^{24})$, as the hidden 1 always exists on both operands. Hence, the result of the multiplier belongs in the range $[2^{46}, 2^{48})$, a 48-bit fixed point integer. The result is treated as a fraction with a decimal point between bit 45 and bit 46. The value of bit 47 determines whether the multiplication of the two fractions being greater than 2 or not. If the result is greater than 2 (i.e., bit 47 is 1) then the mantissa and the exponent of the result are normalized, such that the mantissa is bit 46 to bit 24 of the multiplication. The exponent addition result is increased by 1. From this characterization of the floating point multiplier, we assume that voltage scaling would result in a timing error on the 24-bit multiplication, specifically in the most significant bit of the result, the 47-th bit. Since this critical bit is added to the sum of the two operands' exponent part, if the critical bit is flipped, the result would differ from the correct output by a factor of 2 (i.e., it would either be multiplied or divided by 2).

Thus, we assume the following error model, for timing miss rate p,

$$\text{FP\_MUL} : R = (!p)?\ C\ :\ 2^{\pm 1}C$$

where R denotes the latched result and C denotes the current correct result of the computation. The selection of the factor being 2 or 0.5 is determined whether the critical bit was flipped from 0 to 1, or 1 to 0, as the critical bit being flipped propagates to the result's exponent being added by 1 when it should remain the same, or remained the same when it should be added by 1.

4.2.2 Critical Bits of Floating Point Adder

A 32-bit floating point adder consists of normalization of the operands, so that the mantissa with smaller exponent is aligned with the mantissa with the larger exponent by shifting right the difference of the exponents. The exponent of the result is selected to be the larger exponent of the two operands, and the normalized mantissas is inputted into the 24-bit fixed point adder. The result of the 24-bit fixed point addition is also normalized again such that the result has 1 in its most significant bit, either shifting right or left. (Fig 4.2).

Since a variable length right/left shifter (a barrel shifter) is costly in terms of performance, the process is divided into cases to designate which operands use the first variable-length right shifter for normalization, and which operands use the second variable length left shifter for normalization.

This differentiation is possible as the variable-length left shift at the end of the addition only occurs when the two operands are subtracted in a close range, and these 'close range' operands need not go through a variable-length right shifter before the addition, as the normalization is bounded by either 0 or 1 difference of the operand. We consider three cases: (i) **ADD** : The signs of the two operands are the same, (ii) **SUB-FAR** : The signs are different and the exponent difference of the two operands is bigger than 1, (iii) **SUB-CLOSE** : The signs are different and the exponent difference is either 0 or 1.

In the first case, the larger operand's mantissa (appended with the hidden bit 1) would be a 24-bit fixed point number ranging from $[2^{23}, 2^{24})$, and the smaller mantissa would also be a 24-bit long normalized fixed point number in the range $[0, 2^{24})$. Therefore,

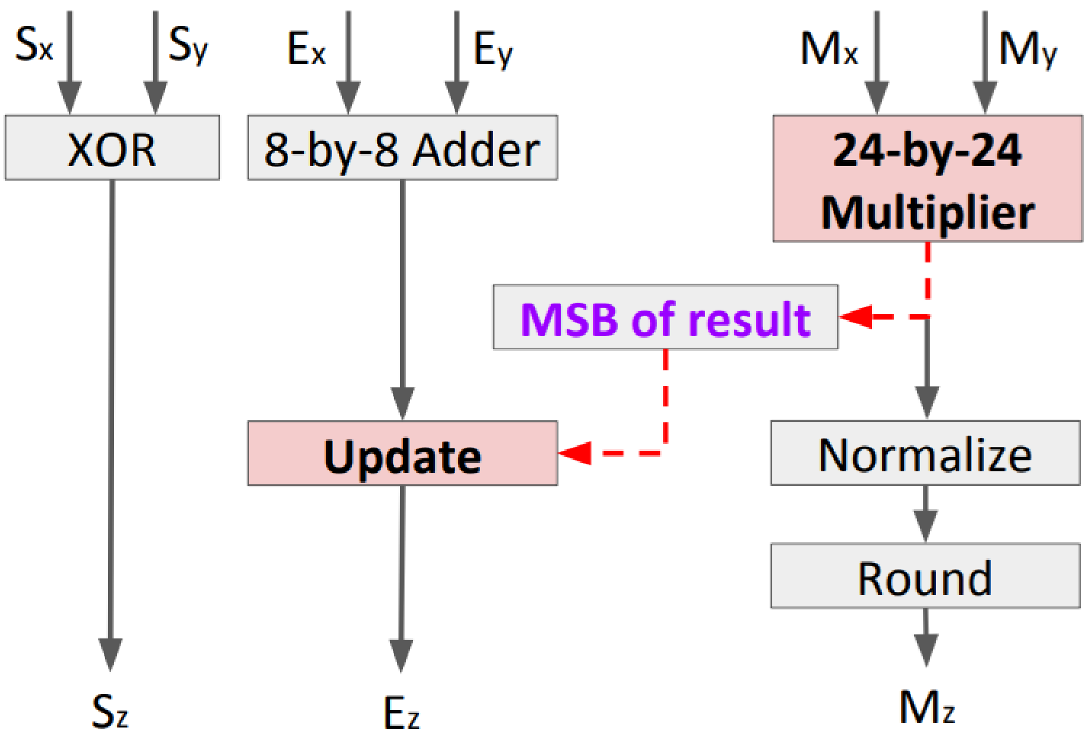

FIGURE 4.2: Floating-point addition architecture

the result of the mantissa addition would be a 24-bit long fixed point number, with a carry-out bit, belonging in range $[2^{23}, 2^{25})$. In this case, we assume the critical path is the computation of the carry out bit of the result, which similarly to the floating point multiplier architecture, is added to the larger exponent of the two operands. Therefore, if there is a timing miss, we assume that the latched exponent is diverging by 1 from the correct exponent, where the direction is dependent on whether the carry out bit has flipped from 0 to 1 or from 1 to 0.

In the second case where the exponent difference is bigger than 1, the less normalized operand of the mantissa subtraction belongs in range: $[0, 2^{22})$, and the result of the 24-bit mantissa subtraction belongs in range: $[2^{22}, 2^{24})$. We assume the critical path is the computation of the 23-rd bit, which is the most significant bit of the subtraction result. If this bit is 1, the result exponent would be the same as the larger exponent, but if the bit is 0, the result exponent would be smaller from the larger exponent by 1.

In the last case, the number of normalization (right shifts) after the subtraction is computed in the LZA (Leading Zero Anticipation) unit, and NLZ (Number of Leading Zeroes) in range $[0, 24)$ shifts the mantissa and subtracts to the larger exponent of operands. We assume the simultaneous critical path is computing the 5 bits consisting of the NLZ, and the divergence of the result is 2^{NLZ} multiplied or divided.

Thus, we assume the following error model, for some timing miss rate p,

$$\begin{aligned}\text{FP\_ADD}: R = &(!p)?\ C :\\ &(S_1 = S_2 \text{ or } |E_1 - E_2| > 1)?\ 2^{\pm 1}C :\\ &2^{\Delta \text{NLZ}}C\end{aligned}$$

where R denotes the latched result, C denotes the current correct result of the computation, S and E denotes the operands' sign and exponent and NLZ denotes the number of leading zeros of the result of mantissa subtraction.

4.2.3 Critical Bit Error Model Behavior Analysis

By using the Critical Bit model, we can explain the bit-wise error behavior of a random trace of FP_ADD and FP_MUL, obtained by b-HiVE and Liu and can characterize the bit-wise error behavior as follows:

- The mantissa bits (bit 0 to bit 51) have uniform probability, increasing from 0% to 50% as voltage decreases.
- The exponent bits (bit 52 to 62) show an exponential decrease as the bit location increases, with a peak at the most significant bit.
- The sign (bit 63) is always correct.

For floating point multiplication, the mantissa's bit-wise uniform error rate can be explained through the critical bit, the most significant bit of the mantissa multiplication, being delayed and flipped. The probability of the critical bit being flipped here would increase as the supply voltage decreases, as different operands would exhibit slightly different delay in the computation of the critical bit, and the flip would propagate to both the exponent being added or subtracted by 1 and the mantissa being shifted left or right by 1. The mantissa being shifted left or right by 1 would result in a bit-wise random mantissa, as there is no bit-wise correlation between the bit values before the shift and after the shift. This also explains the exponential decrease of the exponent bit-wise error rate. As for a higher order bit to flip from an addition or subtraction of 1, all bits before the bit has to be respectively 11..1 or 00..0. Hence, adding or subtracting 1 from a timing error of the critical bit would have have exponentially less impact on the bit-flip rate of higher order bits of the exponent. The peak in the most significant bit of the exponent can be explained by the timing error of the exponent adder fitting inside a single pipeline stage of the floating point addition. We do not consider the timing violation of the addition of the operands' exponents in our *Critical Bit* model, as the adder can be stretched out multiple stages which the mantissa multiplier inevitably goes through.

For floating point addition, the mantissa's bit-wise uniform error rate can be similarly explained as above, except that floating point addition shows more discrepancy between the mantissa's error rate and the exponent's error rate. As a result, the mantissa's uniform error rate increases to 50% while the exponent stays accurate. Since the exponent is accurate, we can infer from Fig 4.2 that the adder works fast enough and correctly, but the L1/R1 shifter, the rounding circuitry, and the multiplexer of the mantissa has a critical path that results to a uniform bit-wise error rate. The exponents exponential decrease of bit-wise error rate can be explained with the critical bit being flipped, equivalent to floating point multiplication. The peak in the most significant bit of the exponent can be explained by the comparator of the operands' exponents, and selecting the larger exponent as a result of the comparison. If the compare result of the exponent is flipped, the smaller exponent would be selected as the result. While this timing error adds bit-wise randomness in the other bits of the exponent, the most significant bit is biased such that the larger exponent has higher probability of 1 and the less exponent has higher probability of 0, which indicates that the most significant bit has higher chance to flip due to value anti-correlation of the most significant bit.

From the above observations, we conclude that *Critical Bit* error model can explain the error behavior of each arithmetic operation in a small, specific range of voltages, i.e. from 0.7V to 0.8V for floating point multiplication and addition.

4.3 Comparing Critical Bit with Other Error Models

We select a few well-known fault-tolerant applications Gaussian Filter, Sobel Operate and Fast Fourier Transform to test how the selection of error model from 6 different options (Critical Bit, b-HiVE, Random, Uniform on the mantissa, Uniform on the exponent, and Uniform on all bit locations) affects the characterization of error tolerance of such benchmark applications. We count how many times each operation of the annotated approximate region returned a different output to calculate the observed error rate. We expect UniformM to be close to b-HiVE (and the actual processor behavior under voltage closer to nominal voltage) for small error rates, UniformE to be very pessimistic as the exponent being different drastically changes the output, and UniformA somewhere in the middle. The output image or signal of each benchmark is collected and compared to the accurate output with the above measurements, and projected with the observed error rate.

Fig 4.3, Fig 4.4 and Fig 4.5 shows the PSNR and error rate of 3 benchmark applications injected with 6 different error models. The defining characteristic of all error models is that PSNR (accuracy) decreases as the error rate increases from 0, which validates each error model as the lower the voltage should have higher error rate and thus less accurate results. We also note that from Fig 4.5, the 3 models Random, UniformE and

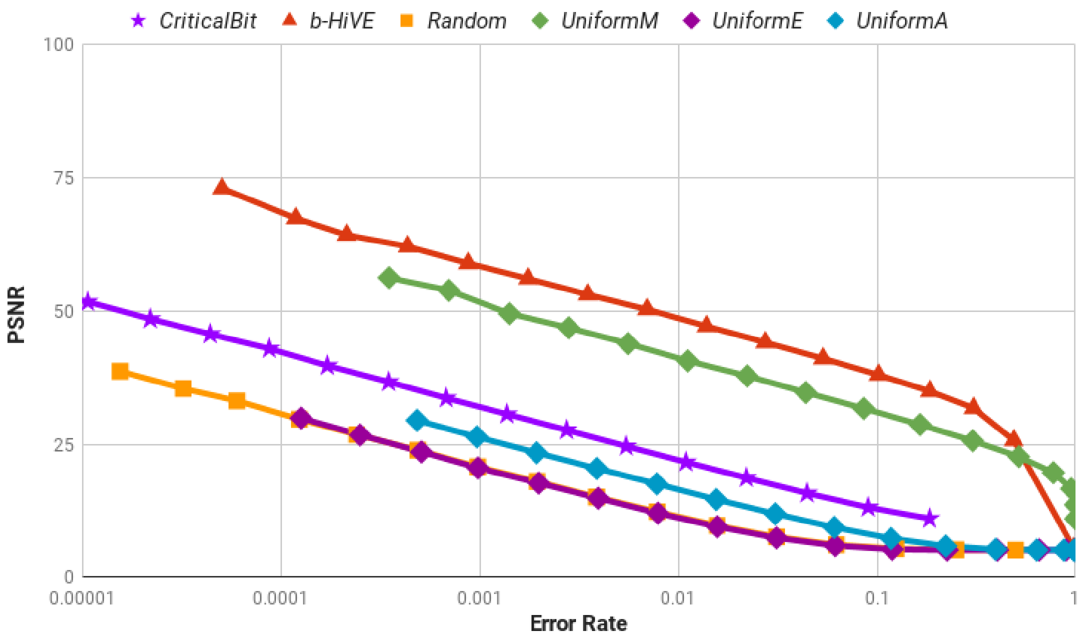

FIGURE 4.3: PSNR(dB) vs. Error Rate of Gaussian filter

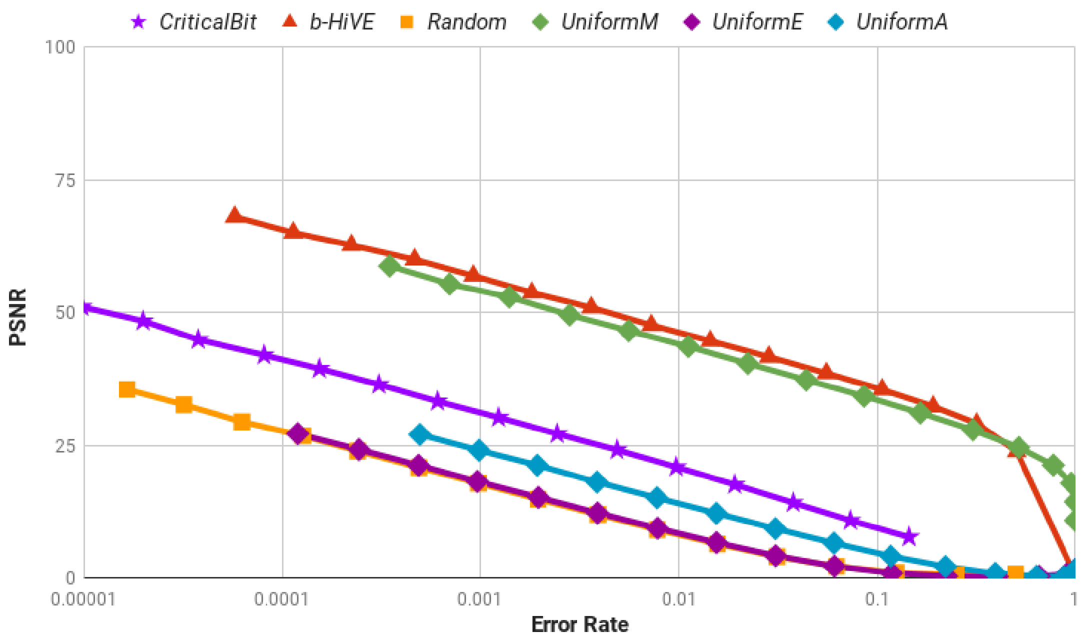

FIGURE 4.4: PSNR(dB) vs. Error Rate of Sobel operator

UniformA are excluded from the graph due to all having outputs of PSNR $= -\infty$ for any error injection rate. This shows that while FFT has a strictly lower error tolerance than Gaussian-filter or Sobel-operator, these 3 error models Random, UniformE and UniformA that change bits outside of the mantissa show significantly different behavior from the other 3 models CriticalBit, b-HiVE and UniformM. The FFT algorithm differs from the Gaussian filter and Sobel in its error tolerance because each of its computations is non-discrete; so errors, if catastrophic, will propagate to all of the output values.

Figure 4.5: PSNR(dB) vs. Error Rate of FFT

Fig 4.3 and Fig 4.4 shows that of the 6 error models, UniformM and the b-HiVE model project the highest accuracy, followed by CriticalBit, UniformA, and then UniformE and Random, both projecting lowest accuracy. The order of accuracy is predictable, since UniformM and b-HiVE are bit-wise mantissa-only (for low error injection rates) independent error models which include the case where the less significant bits of the mantissa are flipped as errors, while other models handle timing violations in the exponent. The CriticalBit is the most accurate from the remaining models, while CriticalBit is bounded to change the output by a ratio of 2, UniformE, Random and UniformA have no limits on changing the exponent, even up to 2^{256} ratios. The UniformA produces more accurate outputs than Random and UniromE as UniformA counts errors that happen only in the mantissa. While the b-HiVE model is close to a UniformM model, it is clearly different from CriticalBit. b-HiVE and UniformM both assume a low bit-wise independent error rate, while CriticalBit assumes a single critical bit flip error rate. Hence, as explained in Section, the discrepancy of the models comes from the assumption of the rounding hardware being the critical path, while CriticalBit assumes that the 24-bit adder and multiplier's result is the critical path. Therefore from the architectural explorations, an adequate FPU architecture that supports fast rounding can assume that the exponent and mantissa shift should be correlated to the critical bit.

FIGURE 4.6: Gaussian-filter output with 6 error models, each image PSNR=20dB

Fig 4.6 shows sample outputs of Gaussian-filter injected with each error model when PSNR of 20dB is selected. While the difference from the original Gaussian-filter output is significant, we can also acknowledge that while CriticalBit, b-HiVE and UniformM (top) produce similar images, Random, UniformA and UniformE (bottom) produce different kind of images, where the glitches are very clear.

We see that each error model shows largely different behavior in each application. For example, when we assume a 0.01 error rate due to voltage scaling, Gaussian-filter can output images of PSNR from 10dB (the lowest) to 50dB (the highest), depending on which error model is applied; when we assume we can only tolerate 30dB PSNR outputs from timing errors due to voltage scaling, the Sobel-operator can tolerate error rates from 0.0001 to 0.3. Since these differences lead to great divergence in estimation of both power savings and performance enhancements from voltage scaling, the selection of error injection model is crucial for correct analysis. For the above reasons, we use the Critical Bit error model for the simulation of our voltage scaling policies.

Chapter 5

Dynamic Voltage Scaling Policy

5.1 Error Model

Many works focusing on error-tolerant and approximate computing follow simplistic error models such as single bit-flip probabilities, uniform distribution models or random values [4, 16, 17] that are not able to fully capture the error behavior of functional units (FUs). For example, the error models considered in EnerJ [16] are basic fault injection error models with single bit flips at the output using uniform distribution, random selection or previously seen values for the output. The work of Esmaeilzadeh et al. [4] relies on uniform bit-error models with set error probabilities per component. In [6] Krimer et al. developed a model for the expected probability of errors due to timing violations when the supply voltage is reduced in integer adders and multipliers, but this model does not take into account bit location and history of computation.

Most of the existing error models do not cover important families of FUs, such as Floating Point Units (FPUs) and bit-wise logic operation units, which have different behavior than integer adders. Moreover, these models ignore value correlation, computation history, and bit-wise error variability. Tziantzioulis et al. [18] introduced b-HiVE, an error model that considers these three factors and showed that the bit-wise error rate of a functional unit's operation can be predicted more accurately. However, b-Hive relies on the independence of bit-wise error rates. Whang et al. [19] address this limitation by tying the error to the critical path in the floating point unit. In particular, the flip rate at each bit location is dependent on the current and previous computation of the signals on the critical path of the computation. These "critical bits" are responsible for the bit flips in the result when the voltage is gradually scaled down. For floating point multiplication, this translates to a potential timing error on the most-significant bit of the mantissa. In other words, if this critical bit is flipped, the result would differ from the correct output by a factor of 2. For floating point addition, this translates to errors in the most-significant bits of the mantissa from the addition/subtraction phase or the adjustment to the exponent needed in the normalization phase. We believe the work of [19] offers the most accurate model for capturing error error behavior in floating point units and therefore we incorporate this model into our implementation and evaluation.

5.2 Non-approximating DVS Policies

On top of the HTM infrastructure described in Section 3.2, we are proposing an approximation-aware error management Dynamic Voltage Scaling (DVS) policy. To evaluate the capability of our approximation-aware policy to save energy and improve performance, we compare it to existing error-management policies, that do not apply approximations and instead correct all occurring errors.

In [13] the authors suggest two such policies on top of an HTM infrastructure: The Point of First Failure (POFF) and Thrifty Uncle/Reckless Nephew (TURN) policies. The POFF policy is a pessimistic approach in which the supply voltage is gradually scaled down in steps until the first failure occurs, in which case the voltage is immediately scaled up a step. Hence, with POFF any detected timing error results in the ongoing transaction being immediately aborted and re-executed after the voltage is increased. This policy leads to execution just above the edge of failure. While the abort rate is minimized and the energy consumption is improved, this approach does not reach the maximum potential in energy savings.

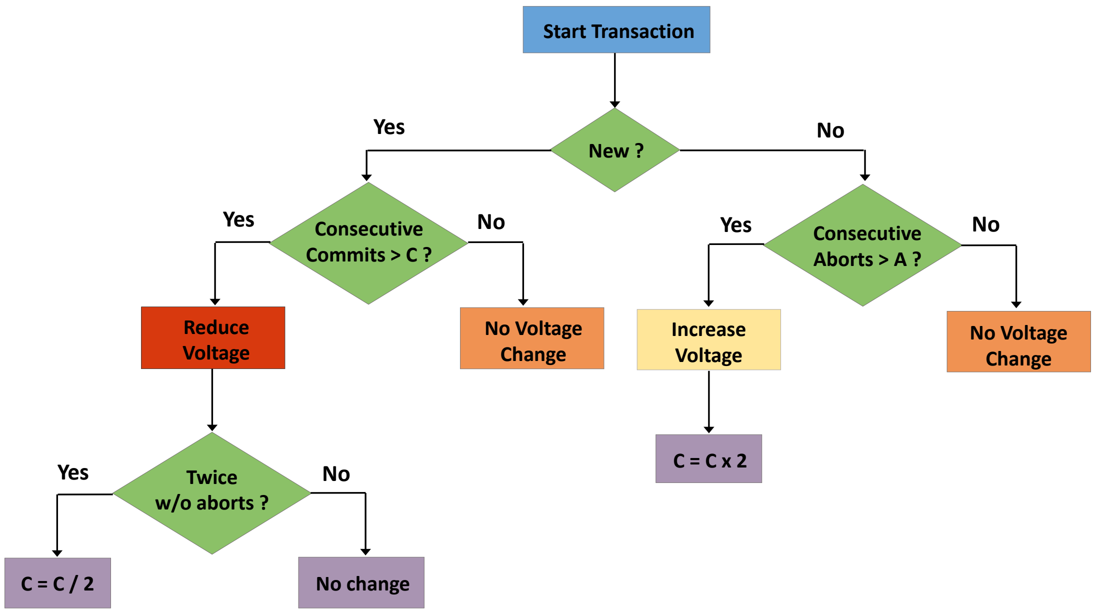

FIGURE 5.1: Flow Diagram of *TURN* DVS Policy [13]

To address these shortcomings, the authors of [13] propose the *Thrifty Uncle-Reckless Nephew* or *TURN* policy. This policy optimistically scales the voltage beyond the point of first failure with the goal to achieve better energy savings. Since the error rate is increased beyond the point of first failure, the transaction abort rate increases as well, which in turn leads to an increase in energy consumption and execution time due to more transactions recoveries and re-executions. This policy thus has two parts:

- The reckless nephew optimistically scales the voltage down for better energy savings, based on the number of consecutive successful transaction commits.
- The thrifty uncle moderates the energy loss from the increased number of aborts (due to over-aggressive voltage scaling), by setting up a threshold based on the number of transaction aborts and commits.

This policy leads to execution beyond the edge of failure and better energy savings. However, the policy still incurs a significant runtime overhead (8%) and it still does not reach the maximum energy saving potential. The flow diagram of the *TURN* policy is described in Figure 5.1

In this paper, we propose an error management scheme that opportunistically ignores errors in order to save more energy. It does so by allowing a certain rate of errors to be left uncorrected and aborting only when the acceptable error threshold has been reached. In the following section, we describe in detail our proposed error management policy called *IgnoreTM.*

5.3 Approximating DVS Policy with *IgnoreTM*

We now consider which instructions can be approximated. For a DVS policy to be able to approximate instructions by ignoring timing errors, consideration of which instruction can be approximated is necessary.

We place instructions into two categories: *"approximatable"* and *"non-approximatable"*. For our experiments, we define approximatable instructions as floating point add, multiply, copy, negate (but not load/store, CPU to/from FPU register moves), and non-approximatable instructions as any other instructions that if approximated would result in significant loss in output quality, hence they must be kept accurate. In general, users should be able to annotate each instruction in their programs where approximation will be applied and those where computations will be kept accurate. Then, on top of that annotation, we use HTM transaction boundaries to only apply voltage scaling under these boundaries and keep all other program parts protected. The program regions that are not approximated must have HTM protection so that if a timing violation occurs there, the HTM support mechanism can correct it. The instructions that are annotated as approximatable and cause timing violations are potentially ignored and left uncorrected. However, these regions are also protected with HTM transaction boundaries, so that if the error threshold tolerance is exceeded, the HTM support can start recovering from timing violations and potentially adjust the voltage level based on the *IgnoreTM* policy.

Voltage adjustments are determined based on the number of completed transactions between aborts. In our *IgnoreTM* policy, we take into account the frequency of a timing violation during a transaction that has to be corrected, and its effect on the total system energy consumption (analyzed in Section 5.4). When such frequency of a timing violation is too high (thus increasing the number of aborts and total energy consumption), we increase our operating voltage in order to save energy by lowering the abort rate. The *IgnoreTM* policy balances ignoring timing violations, correcting timing violations, and changing voltage levels to opimize energy savings while maintaining acceptable program output accuracy.

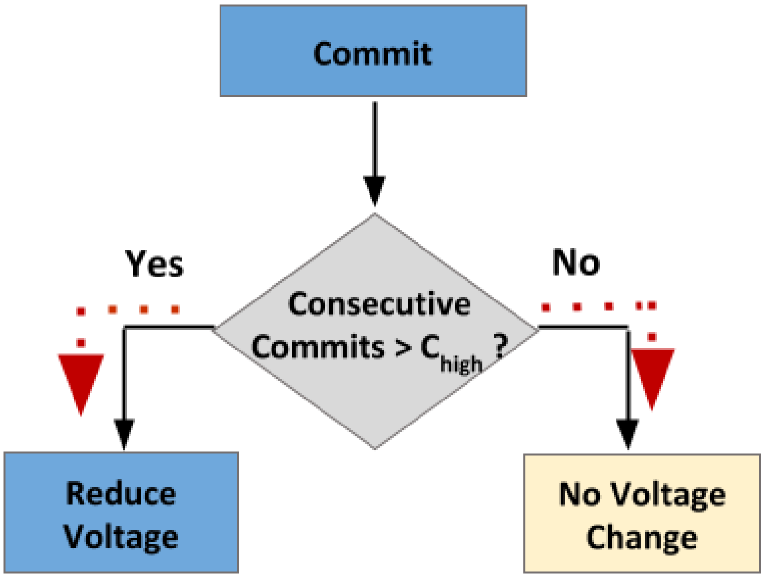

FIGURE 5.2: Flow Diagram of *IgnoreTM* DVS Policy on Commit

Figure 5.2 and 5.3 shows the execution flow of our proposed policy. Here, we discard the notion of consecutive aborts, as we aim to have no transaction with multiple retries. Rather, we set two thresholds for consecutive commits, C_{high} and C_{low}. The C_{high} threshold determines how many consecutive commits are necessary in order to safely scale down the operating voltage by one step. The C_{low} threshold determines how many consecutive commits must happen between two aborts. If this number is not reached, then the policy realizes that the current operating voltage is dangerous and increases the voltage by one step. By aiming for an abort rate between 1% and 10%, we set $C_{high} = 100$ and $C_{low} = 10$, to allow exploring lower voltages when 100 consecutive commits were successful, and retain a higher voltage when the previous abort was less than 10 commits away.

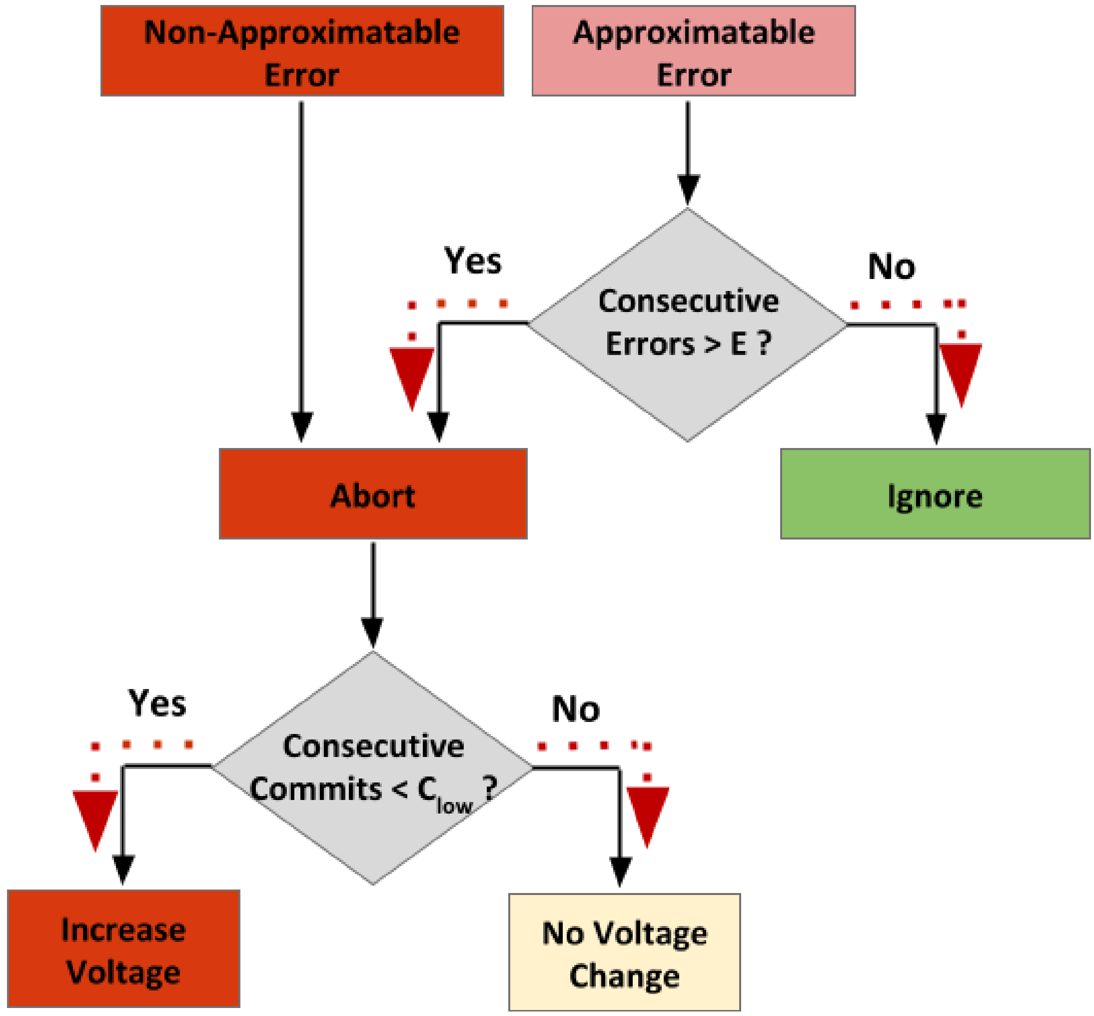

FIGURE 5.3: Flow Diagram of *IgnoreTM* DVS Policy on Timing Violation

We also define another threshold, E, the maximum number of ignored timing errors in a single run of a transaction. E determines the level of approximation the transaction will tolerate. Since every benchmark and every use case of each benchmark can have different error thresholds, we implement E as a user input: a parameter set in by a specially encoded instruction. The user can set a threshold for the number of ignored timing errors *per each transaction*. While running each transaction, *IgnoreTM* will accumulate the number of ignored timing errors, and abort to retry when the threshold E is crossed. Hence, we allow the software to decide E for each transaction, which gives the user some control over the level of expected approximation. We assume that compilers never mark system calls as approximatable, so any timing violation in a system call will be caught and reverted via the HTM recovery mechanism explained below.

The parameters for the *TURN* and *IgnoreTM* policy are summarized below in Table 5.1.

General measures	
CC	# Consecutive successful commits without aborts
CA	# Retries for one transaction
CE	# Ignored timing errors during single run of transaction
POFF	
-	$V\downarrow$ until first timing error
TURN	
C_{turn}	$V\downarrow$ if $CC > C_{turn}$
A_{turn}	$V\uparrow$ if $CA > A_{turn}$
IgnoreTM	
C_{high}	$V\downarrow$ if $CC > C_{high}$
C_{low}	$V\uparrow$ if $CC < C_{low}$
E	Abort transaction if $CE > E$

Table 5.1: Parameters and Summaries of DVS Policies

5.4 Characterizing C_{high} and C_{low}

We now calculate the expected energy consumption per successful transaction and the expected number of consecutive commits based on the scaled voltage level. We then analyze the optimal expected consecutive commits to deterimine C_{high} and C_{low}.

First, let V be the scaled voltage level, V_0 be the nominal voltage, and $v = V/V_0$. Also, let p be the possibility of a timing violation in 1 cycle, N be the fixed number of cycles of the current transaction, $\bar{n}$ be the expected number of cycles up to a successful commit of transaction, and $\bar{c}$ be the expected number of consecutive commits. Here, we assume the scaled voltage level is fixed to V, and there is no additional runtime overhead for retrying a transaction other than the lost computation cycles. Then,

$$\bar{c} = \frac{1}{1-(1-p)^N}, \quad \frac{\partial \bar{c}}{\partial v} = \frac{-N\bar{c}(\bar{c}-1)}{1-p}\frac{\partial p}{\partial v}$$

$$\bar{n} = N(1-p)^N + (1-(1-p)^N)(\bar{n} + p + ... + Np(1-p)^{N-1})$$

$$\bar{n} = \frac{1-(1-p)^N}{p(1-p)^N} = \frac{1}{p(\bar{c}-1)}, \quad \frac{\partial \bar{n}}{\bar{n}\partial v} = (1 - \frac{Np\bar{c}}{1-p})\frac{\partial p}{-p\partial v}$$

Let C be the effective capacitance of the processor (or voltage scaled region), C_0 be the effective capacitance of the remaining regions of the system, $E_0 = C_0V_0^2$, $M = C/C_0$, and $\bar{E}$ be the expected energy consumption per successful transaction. Then,

$$\bar{E} = \bar{n}(C_0V_0^2 + CV^2) = \frac{E_0(1+Mv^2)}{p(\bar{c}-1)}$$

$$\frac{\partial \bar{E}}{\bar{E}\partial v} = \frac{2Mv}{1+Mv^2} + (1 - \frac{Np\bar{c}}{1-p})\frac{\partial p}{-p\partial v}$$

Now, we assume a timing violation probability as exponenetially increasing as voltage decreases (i.e., $\partial p/\partial v = -\beta p$), and for V to be an optimal voltage level, $\partial \bar{E}/\partial v = 0$ has to hold. Therefore,

$$\frac{Np}{1-p}\bar{c} = \frac{2Mv}{\beta(1+Mv^2)} + 1$$

Since $Np/(1-p) = N((\frac{\bar{c}}{\bar{c}-1})^{1/N} - 1) \approx \ln(\frac{\bar{c}}{\bar{c}-1}) \approx \frac{1}{\bar{c}} + \frac{1}{2\bar{c}^2}$, we get

$$\bar{c} \approx \frac{\beta(1+Mv^2)}{4Mv}$$

Here, we can get the soft boundaries of the optimal number of consecutive commits, based on $M = C/C_0$, the effective capacitance ratio between the processor vs. the rest of the system, and β, the timing violation increase factor based on voltage drop. In our system, we characterize $M \approx 1$, and $\beta \approx 50 \ln 10 = 115$ ($10\times$ increase per $0.02V_0$ voltage drop). Last, $v = V/V_0$ is always inbetween 0.7 to 1.0, as going below 70% nominal voltage introduces too many timing violations.

So, we can compute $\bar{c}$ to be near 57 to 61. Since the voltage steps aren't continuous, the expected consecutive commits will change by approximately 10X each voltage increase step, hence we should choose $C_{high}/C_{low} \approx 10$. Therefore, we use $C_{high} = 100$ and $C_{low} = 10$ as our commit threshold parameters. We also characterize the effect of different parameters in Section 6.2.

Note that $\bar{c}$ can be set *without considering the length of the transaction*, which can differ across the program. By $\bar{c}$ being agnostic to transaction length, we can provide a simpler DVS policy with constant C_{high} and C_{low} values and dependent only on β and M.

Chapter 6

Results and Discussion

6.1 Experimental Setup

We use a cycle-level SystemC simulator of a SoC with ARM processors containing floating point units, private instruction caches, a tightly-coupled data memory (TCDM) and a fast interconnect between processors and memory (Figure 3.1). To isolate the benefits of *IgnoreTM*, we run all experiments with one processor core since all cores operate independently from each other with regards to our policy, and as we will see, the total possible energy savings are heavily dependent on the benchmark we are running. We target the processor, including the matching floating point unit, to be voltage-scaled, where the dynamic and static power consumption of each module is characterized by extrapolation from an implementation of the platform in STMicroelectronics 28nm UTB FD-SOI technology under operating points considered in our experiments, and back-annotated in the simulator. Timing errors are generated on a cycle-basis, with exponentially increasing probability as voltage is scaled down. Specific behavior of floating point units under timing violations are implemented using the Critical Bit error model (as described in Section 5.1 and [19]). Support for error correction is done through the HTM infrastructure (as described in Section 3.2).

As benchmarks, we consider floating point implementations of *Gaussian Filter* [3], *Fast Fourier Transform* (FFT) [20] and *Matrix Multiplication* [5]. We chose these applications due to their heavy reliance on floating point computation, mainly addition and multiplication, with inherent tolerance to errors in the output. Also, these applications lack floating point control instructions such as floating point comparisons, which could change the control flow if the comparison was approximated. We analyze their energy savings, runtime overhead, and quality of output loss from various DVS policies. For Gaussian Filter and Matrix Multiplication, we define quality of output as the Peak Signal to Noise Ratio (*PSNR*). For Fast Fourier Transform, we define quality of output as the Average Relative Error (ARE).

6.2 Tuning Commit Parameters

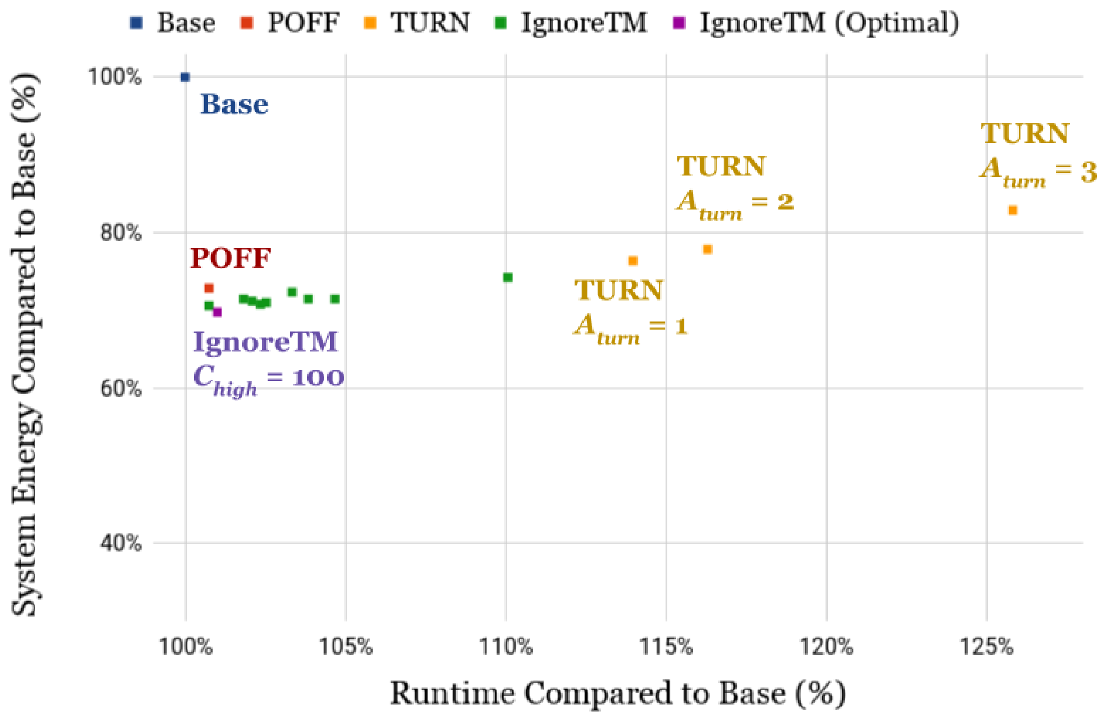

FIGURE 6.1: System Energy Consumption vs. Runtime with different DVS Policies. FFT, No Approximation

As explained in Section 5.2, *TURN* [13] and one of its parameters A_{turn} is a determining factor of how much runtime, system energy, and processor energy can be saved. The same holds for the *IgnoreTM* DVS policy and its parameters C_{high} and C_{low}. Here, we demonstrate this importance by implemeting the following DVS policies on the FFT benchmark:

- Point of First Failure ($POFF$)
- Three versions of *TURN*, $A_{turn} = 3, 2, 1$
- Ten versions of *IgnoreTM*, $(C_{high}, C_{low}) = (20, 2), ..., (200, 20)$ and $E = 0$

In [13], the *TURN* policy used $A_{turn} = 3$ (i.e., more than 3 consecutive timing errors result in a voltage increase). In our experiments in this paper, we also consider increasing the voltage in response to encountering more than 1 or 2 consecutive timing errors, in order to measure the runtime overhead and energy savings when voltage increase is applied more aggressively. Note also that *TURN* in [13] only measured energy in the processor, which resulted in up to 57% energy savings, but since we target the whole

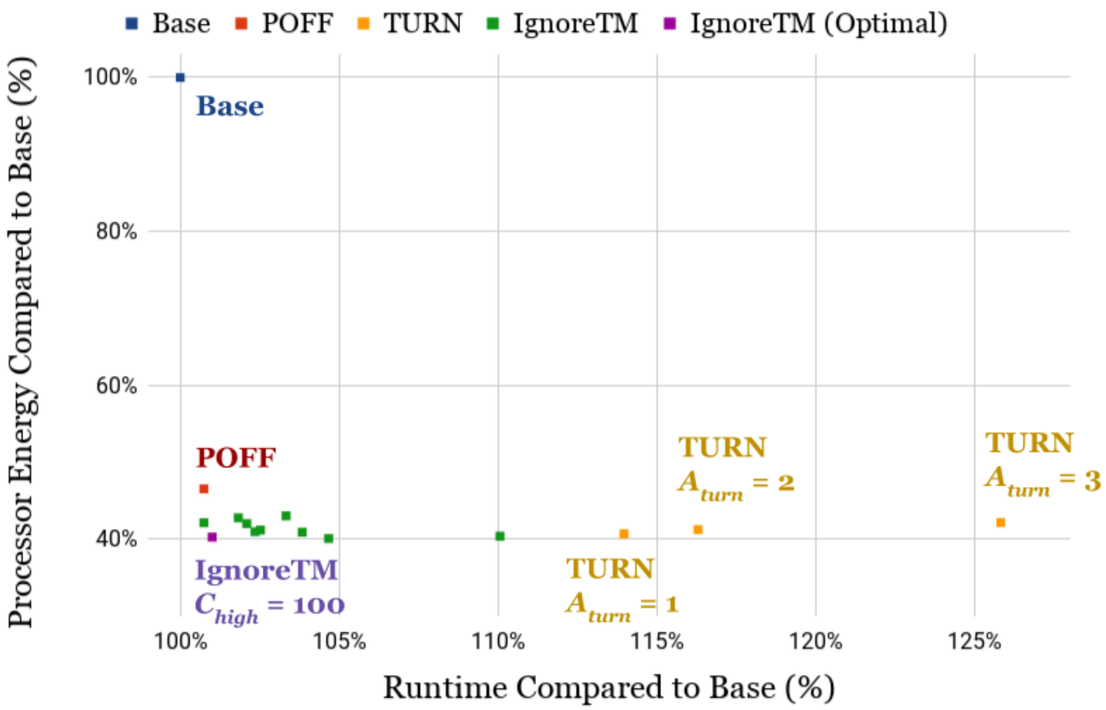

FIGURE 6.2: Processor Energy Consumption vs. Runtime with different DVS Policies. FFT, No Approximation

system energy savings, we reimplement *TURN* to compare full system energy savings of DVS policies.

Figure 6.1 and 6.2 shows the system and processor energy consumption vs. execution time for each policy when running the FFT benchmark. From Figure 6.1, we can see a general trend of energy consumption being proportional to execution time. Here, the *TURN* policy with $A_{turn} = 3$ produces the most runtime and system energy overhead, which is worse than the *POFF* policy that ensures minimal aborts by never reducing the voltage when timing violations start to occur. Also, decreasing A_{turn} from 3 to 2 and 1 for the *TURN* policy shows benefits for both system energy consumption and runtime. The *IgnoreTM* policy being more conservative than the *TURN* policy results in less energy consumption and runtime overhead than *TURN* for all versions. However, the *IgnoreTM* policy does not become more efficient just by increasing C_{high} and C_{low}. There is a point of maximum return where after $C_{high} > 100$ and $C_{low} > 10$, the energy consumption increases without much change in execution time. This is due to the *IgnoreTM* policy becoming too conservative, such that the system does not fully exploit the voltage scaling potential, while still running without any runtime overhead due to no reverts.

From Figure 6.2, we also observe that the processor energy consumption using the *TURN* policy is actually close to optimal, clearly better than *POFF*. This indicates that *TURN* successfully captures the correct voltage level to optimize just the processor energy consumption, but not the whole system energy consumption.

This experiment demonstrates that the we need to consider more conservative voltage levels and fewer abort rates than *TURN* to optimize both runtime and system energy consumption. From the results of this experiment, we choose $(C_{high}, C_{low}) = (100, 10)$ as the tuned commit thresholds that best explain our environment, as analyzed from Section 5.4.

6.3 Tuning Error Tolerance Parameters

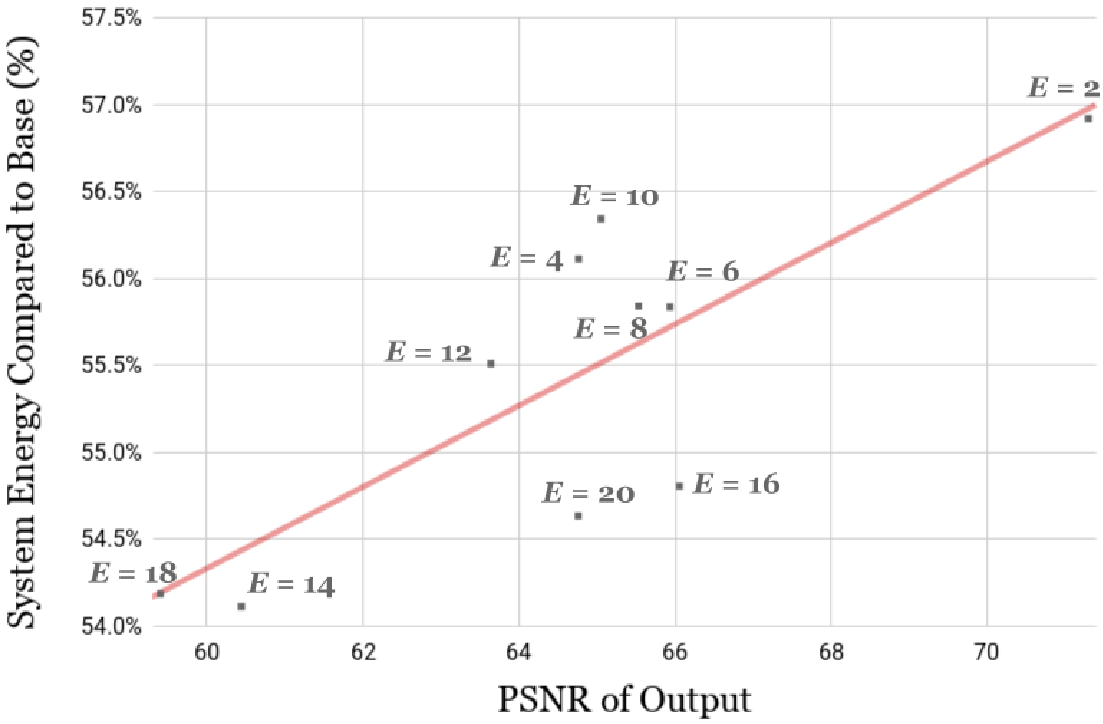

Figure 6.3: System Energy vs. PSNR with different error thresholds. Matrix Multiplication, *IgnoreTM*

Next, we want to analyze the trade-off between accuracy and energy savings. Figure 6.3 shows how energy consumption can be optimized by setting different error thresholds. For these experiments, we choose the matrix multiplication benchmark to show clearly how changing the error threshold E can provide fine-tuned control of the output quality and energy savings. Recall that E is a user-determined threshold of how many timing violations and approximations each transaction can tolerate,

For matrix multiplication with error threshold E ranging from 2 errors per transaction to 20 errors per transaction, we observe that the PSNR of the output can be gradually decreased from 70 to 60, while the energy consumption is also decreased from 57% to 54%. These results demonstrate how the error threshold is a flexible and simple knob for accuracy control that can be used as an input in the start transaction function (i.e., *start_transaction(error_threshold)*), thus providing the user with direct fine-tuned control of both output accuracy level and energy consumption.

6.4 Runtime and Energy Consumption

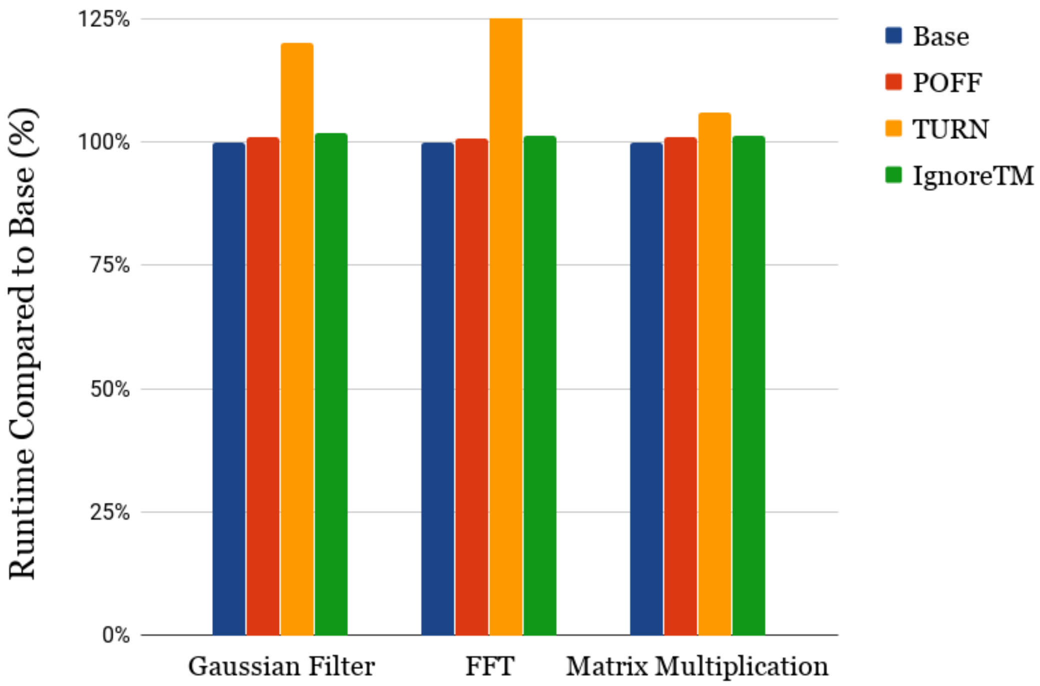

FIGURE 6.4: Runtime of Different Benchmarks and DVS Policies

Given our analysis in Section 5.4, Section 6.2 and Section 6.3, we construct our *IgnoreTM* policy with $C_{high} = 100$, $C_{low} = 10$ to ensure a low enough abort rate. We run three benchmarks: Gaussian Filter, FFT, and Matrix Multiplication, with accuracy thresholds of 30dB PSNR, 25% ARE, and 60dB PSNR, respectively, which in turn determined the error threshold E when running *IgnoreTM* approximations. We also compare our proposed DVS policy *IgnoreTM* against our implementation of previous DVS policies such as *POFF* and *TURN* [13] without approximation.

In Figure 6.4 and 6.5 we show the runtime and energy consumption results for our simulations. These results indicate that *IgnoreTM* can achieve 32% to 45% energy savings (+4% to +15% more compared to the *TURN* technique) with modest runtime overhead of 1.1% to 1.8% relative to no voltage scaling. The additional energy savings come both from being more aggressive with voltage scaling and ignoring errors and from decreasing the number of aborts so that the system has less energy overhead from transaction re-execution. This also explains the significant decrease of runtime overhead compared to the *TURN* policy, as aborts are carefully controlled to remain low.

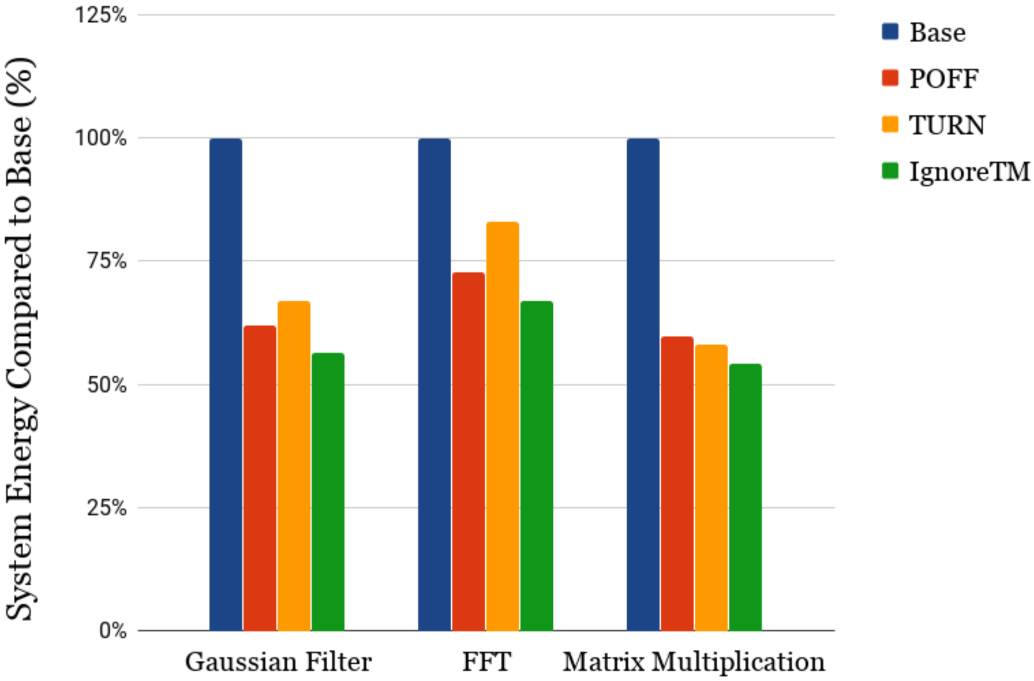

FIGURE 6.5: System Energy Consumption of Different Benchmarks and DVS Policies

Chapter 7

Conclusion

As the supply voltage is scaled down below safe operating margins, timing violations will be triggered. This paper first analyze floating point architecture and propose Critical Bit error model that is used throughout the simulations for timing errors. Using the recovery infrastructure borrowed from Hardware Transactional Memory, we introduce IgnoreTM, a hardware-software environment that supports controlled floating point approximation with significant energy savings and insignificant overhead of circuitry or complexity. Approximation in these scheme is created from allowing timing errors to either be corrected via rollback and reexecution, or ignored if runtime monitors predict the computation error will be sufficiently bounded in the output. As the end goal is to save more energy via voltage scaling without unduly impacting runtime performance or application accuracy, our results show up to 15% additional energy savings compared to always correcting errors, while incurring only a 1% overhead in runtime compared to no voltage scaling. The simplicity of the implementation of HTM and DVS policy (where HTM can be concurrently used for data synchronization), as well as the interface for the programmer to integrate such system in their code to easily control the level of approximation and save energy, brings the novelty of IgnoreTM.

There is more work that can be done towards the investigation of ignoring timing violations. The impact of ignoring timing violations in the processor; in memory; in the whole system can be investigated, and the differentiation of critical and non-critical errors can be more finely defined. The integration of FPUs can be further optimized by having multiple FPUs per processor, or having an approximate version of FPUs instead of the full version - mapping FPU instructions to approximate integer instruction on the hardware level can lead to significant area and energy reduction with minimal accuracy loss. The Critical Bit error model should be further verified, not just by analysis, but by SPICE simulations and/or circuit experiments, which is also our ongoing project. The selection of α and β from the IgnoreTM DVS policy can be further automated for the reduction of preliminary analysis of such hardware voltage scaling.

Bibliography

[1] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter variations and impact on circuits and microarchitecture. In *DAC*, pages 338–342, June 2003.

[2] K. Bowman, J. Tschanz, S. Lu, P. Aseron, M. Khellah, A. Raychowdhury, B. Geuskens, C. Tokunaga, C. Wilkerson, T. Karnik, and V. De. A 45nm resilient microprocessor core for dynamic variation tolerance. *JSSC*, 46(1):194–208, Jan 2011.

[3] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[4] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. Architecture support for disciplined approximate programming. *SIGPLAN Not.*, 47(4):301–312, Mar. 2012.

[5] S. Hong, T. Oguntebi, J. Casper, N. Bronson, C. Kozyrakis, and K. Olukotun. Eigenbench: A simple exploration tool for orthogonal tm characteristics. In *IISWC*, Dec 2010.

[6] E. Krimer, P. Chiang, and M. Erez. Lane decoupling for improving the timing-error resiliency of wide-SIMD architectures. In *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, pages 237–248, June 2012.

[7] E. Krimer, P. Chiang, and M. Erez. Lane decoupling for improving the timing-error resiliency of wide-SIMD architectures. *SIGARCH Comput. Archit. News*, 40(3):237–248, June 2012.

[8] K. Liu. Hardware error rate characterization with below-nominal supply voltages. MS thesis, Northwestern University, 2012.

[9] S. Narayanan, J. Sartori, R. Kumar, and D. L. Jones. Scalable stochastic processors. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, pages 335–338, 3001 Leuven, Belgium, Belgium, 2010. European Design and Automation Association.

[10] K. Nepal, S. Hashemi, H. Tann, R. I. Bahar, and S. Reda. Automated high-level generation of low-power approximate computing circuits. *IEEE Transactions on Emerging Topics in Computing*, PP(99):1–1, 2016.

[11] K. Nepal, Y. Li, R. I. Bahar, and S. Reda. Abacus: A technique for automated behavioral synthesis of approximate computing circuits. In *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '14, pages 361:1–361:6. European Design and Automation Association, 2014.

[12] K. Nepal, H. Tann, S. Hashemi, R. I. Bahar, and S. Reda. The ABACUS tool. software release, 2016. https://github.com/scale-lab/ABACUS.

[13] D. Papagiannopoulou, A. Marongiu, T. Moreshet, M. Herlihy, and R. I. Bahar. Edge-TM: Exploiting transactional memory for error tolerance and energy efficiency. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):153:1–153:18, Sept. 2017.

[14] A. Rahimi, A. Marongiu, R. K. Gupta, and L. Benini. A variability-aware openmp environment for efficient execution of accuracy-configurable computation on shared-fpu processor clusters. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2013 International Conference on*, pages 1–10, Sept 2013.

[15] D. Rossi, F. Conti, A. Marongiu, A. Pullini, I. Loi, M. Gautschi, G. Tagliavini, A. Capotondi, P. Flatresse, and L. Benini. Pulp: A parallel ultra low power platform for next generation iot applications. In *2015 IEEE Hot Chips 27 Symposium (HCS)*, pages 1–39, Aug 2015.

[16] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. EnerJ: Approximate data types for safe and general low-power computation. In *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pages 164–174, New York, NY, USA, 2011. ACM.

[17] J. Sartori, J. Sloan, and R. Kumar. Stochastic computing: Embracing errors in architecture and design of processors and applications. In *2011 Proceedings of the 14th International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*, pages 135–144, Oct 2011.

[18] G. Tziantzioulis, A. M. Gok, S. M. Faisal, N. Hardavellas, S. Ogrenci-Memik, and S. Parthasarathy. b-HiVE: A bit-level history-based error model with value correlation for voltage-scaled integer and floating point units. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2015.

[19] S. Whang, T. Rachford, D. Papgiannopoulou, T. Moreshet, and R. I. Bahar. Evaluating critical bits in arithmetic operations due to timing violations. In *IEEE High Performance Extreme Computing Conference*, Sept. 2017.

[20] A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, and P. Lotfi-Kamran. Axbench: A multiplatform benchmark suite for approximate computing. *IEEE Design & Test*, 34(2):60–68, April 2017.