Evaluating Critical Bits in Arithmetic Operations due to Timing Violations

Sungseob Whang[†], Tymani Rachford[†], Dimitra Papagiannopoulou[†], Tali Moreshet[*], R. Iris Bahar[†]
[†]School of Engineering, ECE group, Brown University, Providence, RI 02912
[*]Dept. of Electrical and Computer Engineering, Boston University, Boston, MA 02215

***Abstract*—Various error models are being used in simulation of voltage-scaled arithmetic units to examine application-level tolerance of timing violations. The selection of an error model needs further consideration, as differences in error models drastically affect the performance of the application. Specifically, floating point arithmetic units (FPUs) have architectural characteristics that characterize its behavior. We examine the architecture of FPUs and design a new error model, which we call *Critical Bit*. We run selected benchmark applications with *Critical Bit* and other widely used error injection models to demonstrate the differences.**

I. Introduction

Transistor scaling has left devices more susceptible to the effects of static and dynamic variability. To ensure reliable operation designers have conservatively added safety margins (guardbands) to the system's operating frequency or voltage, which results in wasted energy and degraded performance. On the other hand, removing guardbands can lead to intermittent timing errors (i.e. signal values due to signals not meeting their timing constraints). Intermittent timing errors begin to appear when the operating conditions (**F**requency, **V**oltage, and **T**emperature) approach the *point of first failure* (PoFF). Beyond that point (e.g., decreasing the voltage further), errors become gradually more frequent and the system's behavior can be coarsely modeled with a probability (frequency) of errors as a function of (F,V,T) [12]. To endure reliable operation many works have proposed error detection and correction techniques at the software and the hardware level [10], [4], [16], [8], [22], [20], [6], [12], [5]. However, maintaining fault-free operation without adding considerable overheads on performance and energy is not trivial.

Approximate computing has emerged as a promising solution to these dilemmas. Approximate computing is based on the observation that exact computation and perfect accuracy are not always necessary. Thus, intentionally ignoring a small percentage of timing errors in specific application regions, would allow for voltage scaling and consequently to energy savings. Indeed, while some application domains require always correct computation and any loss of accuracy can be damaging and cause catastrophic failure (e.g., security applications), there are many other applications (e.g. machine learning, signal processing, image processing, scientific computing, data mining and data analysis) that have an inherent tolerance to inaccuracy and errors. For those applications, trading accuracy with inexact computation can prove beneficial in terms of both performance and energy consumption. Nevertheless, approximate computing poses some key challenges since it requires very careful selection of the code and data portions that can be approximated. A poor choice of the application portions for approximation can lead to unacceptable quality loss on the output result.

Various techniques have been proposed for approximation both in the hardware and the software level [25], [18], [3], [7], [11], [2], [1], [24]. However, many of these works follow simplistic error models such as single bit-flip probabilities, uniform distribution models or random values [11], [25], [26] that are not able to fully capture the error behavior of functional units (FUs). These 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, most existing error models ignore three important factors: i) value correlation, ii) computation history and iii) bit-wise error variability [27]. The authors of b-HiVE constructed 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 when taking these factors into account.

In this paper, we introduce a new error model, called *Critical Bit* model. Fully acknowledging the importance of computation history and value correlation from b-HiVE's approach, this new error model integrates bit-wise dependency by identifying the *critical bit* for each FU architecture. The critical bit corresponds to a signal that is along the critical path of a particular arithmetic operation. We explore the error tolerance of various applications under voltage overscaling, testing various timing error models (including the *Critical Bit* model) across a range of error rates and voltage levels, and evaluate the degradation and accuracy loss that is experienced in the output.

Our results indicate that the *Critical Bit* model and other selected error models produce significantly different behavior on tested applications. This is important since it can lead to very different assumptions about error tolerance of a particular application. These results motivate future use of this model to manage voltage scaling of floating point units to more reliably exploit error tolerance.

II. Background

Static and dynamic variability lead to intermittent timing errors that can be activated or de-activated by voltage, frequency or temperature fluctuations. Intermittent timing errors manifest as timing violations on the processor's critical paths; as the voltage is scaled down, errors initially emerge at low rates that later increase exponentially as the voltage is lowered further. For applications that require 100% accuracy, these types of errors would be unacceptable. However, many application domains, including video, signal and image processing, machine learning, computer vision, data mining/analysis and gaming, are inherently tolerant to inaccuracy and have an intrinsic resilience to errors. In such types of applications, we can identify code regions where intermittent timing errors could be acceptable, if that allows for voltage scaling and energy savings. However, an accurate error model is needed to explore the trade-off between loss of accuracy and energy savings.

While other works have considered approximation through voltage scaling, they employ simplistic error models that are not highly accurate and do not provide details on how these models where produced [14], [28], [25], [11]. Samson et

978-1-5386-3472-1/17/$31.00 ©2017 IEEE

al. [25] proposed EnerJ, an approximate programming model that extends Java with approximate data types that allow approximate variables to be mapped to low-power storage and operations. They introduce approximate-aware ISA extensions to support approximate instructions that are executed on special, voltage-scaled function units (FUs). The error models considered in this work are basic fault injection error models with single bit flips at the output using uniform distribution, or select random or previously seen values for the output. Esmaeilzadeh et al. [11] introduce an ISA extension that allows the compiler to convey what can be approximated and propose a microarchitecture design that supports these extensions and allows dual voltage operation on its components: high voltage for precise operations and low voltage for approximates ones. However, this work relies on uniform bit-error models with set error probabilities per component. Krimer et al. [15] 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.

However, in real applications, consecutive operations often have similar values, thus missing timing does not necessarily result in an incorrect result. At the same time, the history of prior computations often affects the current computation results, since switching a signal is more likely to result in an timing error than keeping the value constant. Finally, the error behavior can vary among different bit locations significantly. Current error models ignore this variability and apply uniform single error probabilities to all bit locations. Tziantzioulis et al. [27] introduced b-HiVE, an error model that achieves 6-10x higher accuracy compared to existing error models, by taking into account these important factors in modeling the error probability of different FUs versus supply voltage: the effect of value correlation between consecutive operations, the impact of computation history and the error rate variation among different bit locations. Liu [17] provided an experimental behavior of arithmetic units under below-nominal supply voltages, using this error model.

In this paper, we introduce a new model, the *Critical Bit model*. Our model still considers computation history and value correlation as b-HiVe does, but it has an important difference. While b-HiVe relies on the independence of bit-wise error rate, the *Critical Bit model* integrates bit-wise dependency by identifying the *critical bit* for each FU architecture, i.e., an intermediate bit/wire that requires significantly longer delay than other intermediate bits. We believe that this model is capable of better capturing the delay behavior of different FPU architectures.

III. Implementation

A. Simulated Error Models of FP Units

To test our benchmarks tolerance to errors, we consider a variety of error models from earlier work [25], [26], [15], [27], as well as our *Critical Bit* model. These models are presented in more detail next.

1) ***Random model****:* EnerJ [25] introduced 3 widely used error injection models: *Random*, *Previous* and *Single* [26], [27]. 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.

2) ***Uniform model****:* b-HiVE's interpretation of Lane-Decouple's model [15], [27] used a uniform error model, where each bit of the result is independently flipped for a bit-wise uniform error rate p. Rahimi [23] 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.

3) ***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, [27] and [17] 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) ***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.

B. FPU architecture and Critical bits selection

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 FP_ADD (floating-point addition) and FP_MUL (floating-point multiplication).

1) ***Floating point multiplication (FP_MUL)****:* 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

978-1-5386-3472-1/17/$31.00 ©2017 IEEE

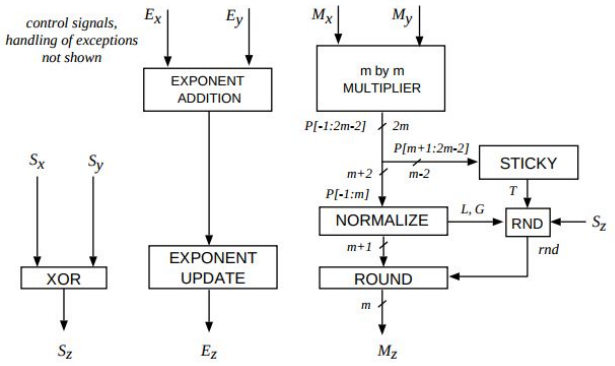

Fig. 1. Floating-point multiplication architecture [9]

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 1).

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.

*2) **Floating point addition (FP_ADD)** :* 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 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

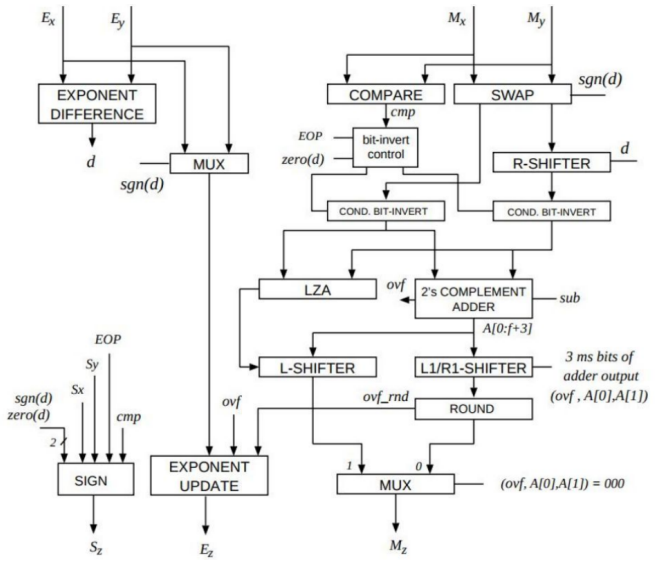

Fig. 2. Floating-point addition architecture [9]

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 normaliztion 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, 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

978-1-5386-3472-1/17/$31.00 ©2017 IEEE

miss rate p,

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

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.

*3) **The Critical Bit model behavior**:* 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 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

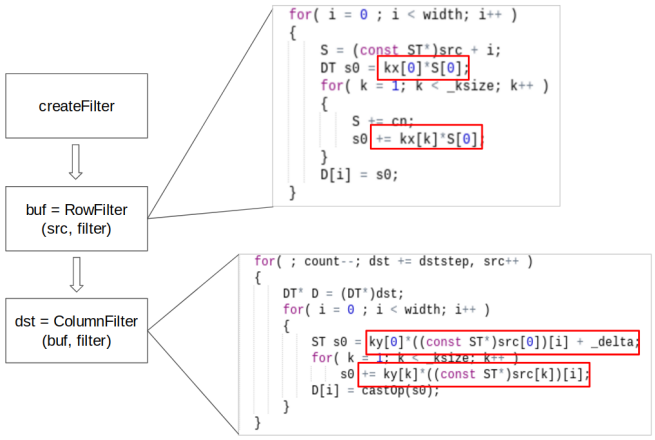

Fig. 3. Approximate region of Gaussian-filter and Sobel-operator

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.

C. Benchmark applications and approximate regions

For any application to benefit from approximation, the selection of approximate regions in the source code is critical, as most code regions require accurate computation for the application to function correctly. For each application we ran, we explain next, how the runtime is divided based on profiling and how we select the approximate regions.

*1) **Gaussian-filter**:* Gaussian filtering takes in an image, defined by a 2-dimensional matrix containing pixel values (usually in range $[0, 255]$), and outputs a smoothed image. The smoothed image has an inherent tolerance to inaccuracy. This is because the Gaussian function has fractions of code computing the exponent coefficients of the filter and hence the output image. Also, the application is usually used to reduce the noise of an input image, which is inherently an approximation.

*2) **Sobel-operator**:* Sobel operator takes in an image, and outputs a (x- or y-)derivative of the image by convolving a derivative filter matrix, usually [-1,-2,-1; 0,0,0; +1,+2,+1] for y-derivative and the transpose for the x-derivative. To test the error models we used the Gaussian filter and Sobel operator from OpenCV [19]. We observed that both applications take up to 99% of the runtime on functions *RowFilter* and *ColumnFilter*.

Fig 3 describes how both the Gaussian filer and Sobel operator applications call RowFilter and ColumnFilter, and which parts in these functions are selected to be approximated. RowFilter and ColumnFilter convolves a pre-computed 1-D filter matrix onto a image matrix. The convolution is first done on the input image structure on each row, and the intermediate result is convolved on each column. The convolution traverses the input matrix, and for each pixel position, sums the multiplication of the coefficient of the filter matrix and the pixel offsetted from the current pixel. The resulting sum is

978-1-5386-3472-1/17/$31.00 ©2017 IEEE

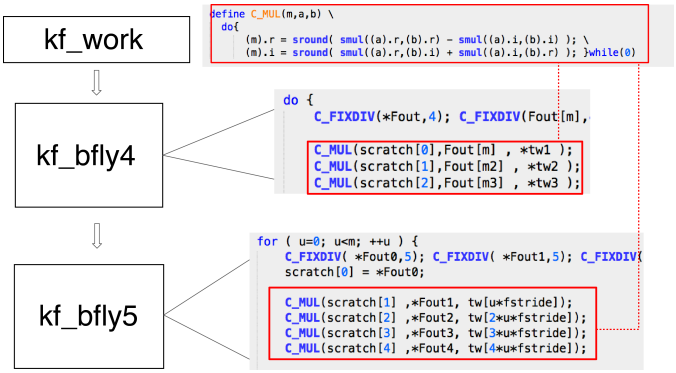

Fig. 4. Approximate region of Fast Fourier Transform

saved in the output. Hence, the multiplication of coefficients and the addition of the multiplication results are the two operations selected for approximation.

*3) **Fast Fourier Transform**:* FFT takes signals mapped in the time domain and transforms them into frequency domain by using the Discrete Fourier Transform (DFT) function:

$$X_n = \sum_{n=0}^{N-1} x_N e^{-i2\pi kn/N} \ (k = 0, 1...N-1)$$

To test the error models we used Fast Fourier Transform from KissFFT [13]. In Fig 4 we show the complex multiply, CMUL, which is where we inject errors, and the two main functions, kf_bly4 and kf_bly5, where CMUL is used. When we profiled our program we found that 77.9% of the runtime is used for allocating memory, and that 16.8% is used for kf_bly4 and kf_bly5 and similar functions, which indicated that the majority of the computation is being done in the complex multiplies with CMUL.

We select complex multiplication from computing $e^{-i2\pi kn/N}$ and x_n as approximate regions, which consists of both floating point addition and multiplication, since this complex multiplication is responsible for the computation of the amplitudes for the set range of frequencies.

IV. Results

Each benchmark application needs a standard measurement of how the result differs from the anticipated (correct) result. We select PSNR (Peak Signal to Noise Ratio) of the error propagated output compared to the sanitary output to determine the difference.

$$\text{PSNR} = 10 \cdot \log_2(\frac{\text{Peak Signal}^2}{\text{MSE}})$$

PSNR is measured in decibels (dB), and larger PSNR indicates higher accuracy.

We test the benchmark applications by running a sample image or signal with 6 error models and various injection rates. The error models include Critical Bit (Critical), b-HiVE (b-Hive), and Random error bit injection (Random). We implement the Uniform model on 3 different boundaries: only the mantissa (UniformM), only the exponent (UniformE) and all bit locations (UniformA), and examine the different impact on PSNR. 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

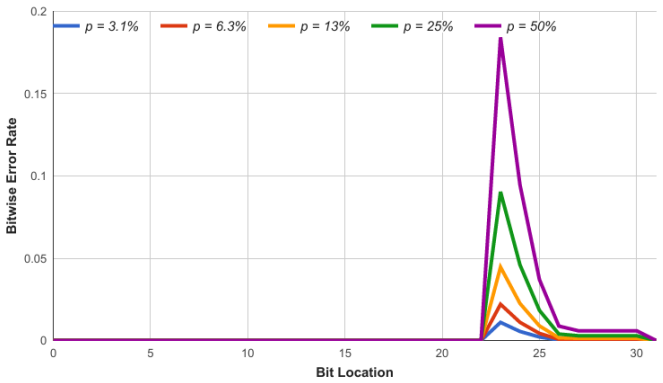

Fig. 5. Observed bit-wise error rate of *Critical Bit* model injected Gaussian-filter

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 5 shows the observed bit-wise error rate of *Critical Bit* model injected Gaussian-filter with different injection rates p, where p ranged from 3.1% to 50%. We observe that the error model produces an exponential decrease of the exponent bits, and the effective error rate is smaller than the injection rate due to consideration of previous critical bit's correlation. Fig 6 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. Fig 7 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 also note that from Fig 6, for FFT, the 3 models Random, UniformE and 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.

Fig 6 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

978-1-5386-3472-1/17/$31.00 ©2017 IEEE

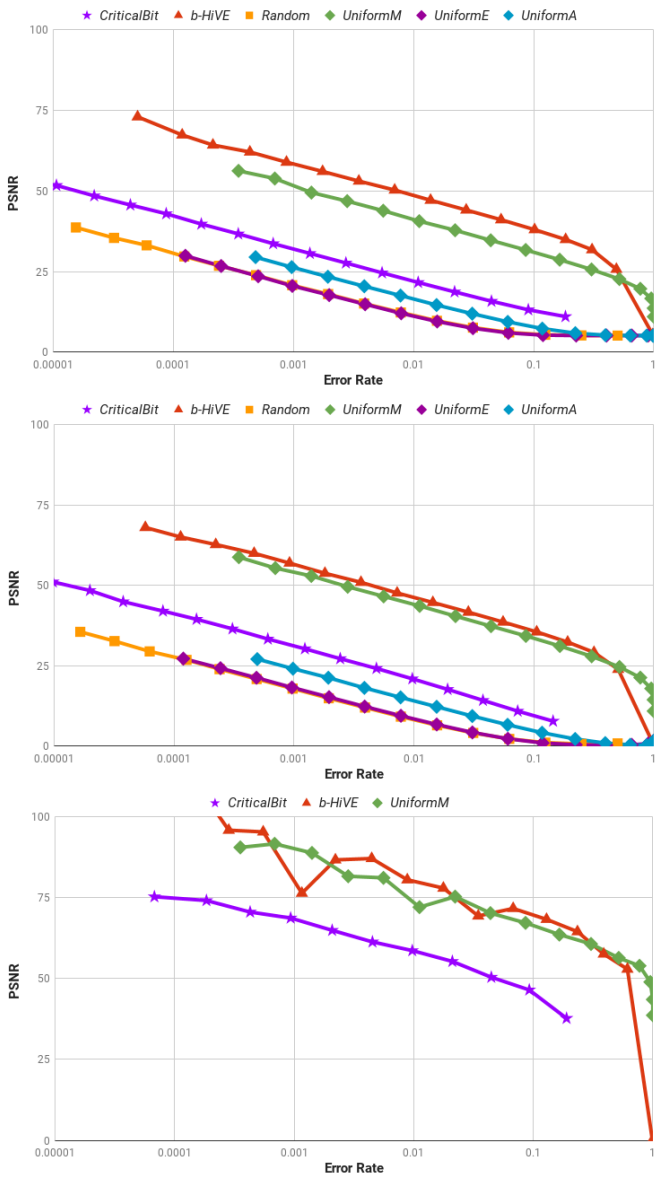

Fig. 6. PSNR(dB) vs. Error Rate of Gaussian filter(top), Sobel operator (middle), FFT (bottom)

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 III-B3, 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.

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

Fig. 7. Gaussian-filter output with 6 error models, each image PSNR=20dB

(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.

V. Conclusions

Allowing timing-induced errors to go uncorrected in floating point applications relies on having accurate knowledge of the application's inherent error tolerance. The selection of the error model is therefore critically important in evaluating the behavior of such applications. In this paper, we presented the *Critical Bit* error model, which can be derived from architectural implementations of floating point arithmetic units, and can be used to further explain prior work's experimental results. We believe this new model more accurately captures erroneous output behavior compared to prior models.

For future work, we plan to synthesis the FP unit in hardware in order to further verify the accuracy of our *Critical Bit* error model. The hardware implementation may also be extended to a pipelined unit to evaluate its affect on timing errors. In addition, we intend to incorporate this error model into a multi-core architectural-level simulator and use it for managing voltage scaling in order to optimally trade off power savings and accuracy of the application. More specifically, for the system to have controlled approximate behavior, error detection and correction mechanisms such as [20], and [21] will be considered.

VI. Acknowledgements

This work is supported in part by NSF under Grants CSR-1319095 and CSR-1519576. We are thankful to our colleagues Andrea Marongiu, Maurice Herlihy, and Jiwon Choe who provided expertise that greatly assisted the research.

References

[1] A. Agarwal, M. Rinard, S. Sidiroglou, S. Misailovic, and H. Hoffmann. Using code perforation to improve performance, reduce energy consumption, and respond to failures. Technical Report MIT-CSAIL-TR-2009-042, MIT, Mar. 2009.

[2] C. Alvarez, J. Corbal, and M. Valero. Fuzzy memoization for floating-point multimedia applications. *IEEE Transactions on Computers*, 54(7):922–927, July 2005.

[3] W. Baek and T. M. Chilimbi. Green: A framework for supporting energy-conscious programming using controlled approximation. *SIGPLAN Not.*, 45(6):198–209, June 2010.

[4] 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.

[5] S. Das, D. Roberts, S. Lee, S. Pant, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. A self-tuning DVS processor using delay-error detection and correction. *IEEE JSSC*, 41(4):792–804, April 2006.

[6] S. Das, C. Tokunaga, S. Pant, W.-H. Ma, S. Kalaiselvan, K. Lai, D. Bull, and D. Blaauw. RazorII: In situ error detection and correction for PVT and SER tolerance. *Solid-State Circuits, IEEE Journal of*, 44(1):32–48, Jan 2009.

[7] M. de Kruijf, S. Nomura, and K. Sankaralingam. Relax: An architectural framework for software recovery of hardware faults. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 497–508, New York, NY, USA, 2010. ACM.

[8] S. Dighe, S. Vangal, P. Aseron, S. Kumar, T. Jacob, K. Bowman, J. Howard, J. Tschanz, V. Erraguntla, N. Borkar, V. De, and S. Borkar. Within-die variation-aware dynamic-voltage-frequency-scaling with optimal core allocation and thread hopping for the 80-core TeraFLOPS processor. *JSSC*, 46(1):184–193, Jan 2011.

[9] M. Ercegovac and T.Lang. *Digital Arithmetic*. Morgan Kaufmann, 2003.

[10] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: A low-power pipeline based on circuit-level timing speculation. In *IEEE/ACM MICRO*, pages 7–, 2003.

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

[12] M. Fojtik, D. Fick, Y. Kim, N. Pinckney, D. Harris, D. Blaauw, and D. Sylvester. Bubble Razor: Eliminating timing margins in an ARM Cortex-M3 processor in 45 nm CMOS using architecturally independent error detection and correction. *Solid-State Circuits, IEEE Journal of*, 48(1):66–81, Jan 2013.

[13] KissFFT API for "keep it simple, stupid" fast fourier transform. https://sourceforge.net/projects/kissfft/.

[14] P. Krause and I. Polian. Adaptive voltage over-scaling for resilient applications. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pages 1–6, March 2011.

[15] 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.

[16] L. Leem, H. Cho, J. Bau, Q. Jacobson, and S. Mitra. ERSA: Error resilient system architecture for probabilistic applications. In *DATE*, pages 1560–1565, March 2010.

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

[18] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn. Flikker: Saving DRAM refresh-power through critical data partitioning. *SIGPLAN Not.*, 46(3):213–224, Mar. 2011.

[19] OpenCV API for computer vision and machine learning, version 2.4.13. http://www.opencv.org.

[20] D. Papagiannopoulou, A. Marongiu, T. Moreshet, L. Benini, M. Herlihy, and I. Bahar. Playing with fire: Transactional memory revisited for error-resilient and energy-efficient MPSoC execution. In *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI*, GLSVLSI '15, pages 9–14, New York, NY, USA, 2015. ACM.

[21] D. Papagiannopoulou, A. Marongiu, T. Moreshet, M. Herlihy, and R. I. Bahar. Edge-tm: Exploiting transactional memory for error tolerance and energy efficiency. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2017 International Conference on*, pages 1–19, July 2017.

[22] A. Rahimi, D. Cesarini, A. Marongiu, R. K. Gupta, and L. Benini. Improving resilience to timing errors by exposing variability effects to software in tightly-coupled processor clusters. *JETCAS*, 4(2):216–229, 2014.

[23] 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.

[24] M. Rinard. Probabilistic accuracy bounds for fault-tolerant computations that discard tasks. In *Proceedings of the 20th Annual International Conference on Supercomputing*, ICS '06, pages 324–334, New York, NY, USA, 2006. ACM.

[25] 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.

[26] 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.

[27] 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.

[28] G. Varatkar and N. Shanbhag. Error-resilient motion estimation architecture. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 16(10):1399–1412, Oct 2008.

978-1-5386-3472-1/17/$31.00 ©2017 IEEE