Lightweight Detection and Recovery Mechanisms to Extend Algorithm Resiliency in Noisy Computation

Sek Chai, David Zhang
SRI International, Princeton, NJ

Jingwen Leng, Vijay Janapa Reddi
University of Texas at Austin

***Abstract*— The intrinsic robustness of an algorithm and architecture depends highly on the combined ability tolerate noise. In this paper, we present an alternative approach for energy reduction for near threshold computing based on a statistical modeling of computational noise induced from noisy memory and non-ideal interconnects. We present this approach as a complement to the standard approximate computing approaches. We show results of the lightweight error checks and recovery based on several design considerations on data value speculation.**

***Index Terms*—Approximate computing, noise resiliency, computation noise, near threshold computing**

I. Introduction

Research has shown that the most power efficient region of operation for a circuit is at near its threshold voltage, and that lowering supply voltage, possibly beyond safe-operation region is a viable option to reduce power consumption [1,2]. This NTV (near threshold voltage) operating region is beneficial for power efficiency due to low voltage swings and quick logic transition. However, circuits operating in NTV may not be deterministic as state transitions may occur due to current noise or transistor mismatches.

In this paper, we advocate an alternative approach that considers the statistical distribution rather than the instantaneous occurrence of the error or noise. That is, certain level noise *can and should be acceptable* as part of the normal operating region, especially when an algorithm can safely operate at that noise level. In this approach, we do not have to detect and correct for hardware errors at every instance, but instead, we check for the statistical distribution and noise levels to be within tolerable levels. For instance, we can lower operating voltages at various levels to reduce power while allowing algorithms to operate in different levels of hardware errors. We introduce the notion of a "computation noise" which is analogous to "sensor noise", whereby the processor and memory subsystem introduces errors in the form of noise that the algorithms can tolerate.

Traditional methods for error recovery such as checkpointing (re-compute if there is an error) are often a computationally and power expensive proposition. Other hardware mechanisms such as ECC (error correcting codes) also has high overhead, especially if we purposefully operate in regions with higher error levels. While previous studies rely explicitly on the resiliency of an algorithm [3] or new circuits/architecture to support NTV [1] or approximate computing [2], they do not explore how algorithms would operate in different levels of detected errors. More specifically, while we understand that lowering voltages can reduce power and raising them when the severity of detected errors exceeds a threshold, there has not been a thorough exploration on how algorithms can operate beyond the "safe-operating region".

By studying algorithmic performance based on probability of detected errors, we can define the range of voltage settings that can be used based on the resiliency of the algorithms. That is, we first quantify regions of operation whereby safe-operation is defined by negligible levels of errors. Naturally if the voltage continues to drop, algorithm performance would degrade at a rate dependent on its construction. In this region of operation, we offer a number of lightweight architectural mechanisms that maintains its algorithmic performance. These lightweight mechanisms range from the simple bit-parity check and simple use-last-copy buffer.

In this paper, we focus specifically on lightweight mechanisms that detect and correct for memory bit-flip errors. Our paper contributions include: 1) a method to dynamically inject computation noise into specified regions of the compiled binary, 2) a set of architectures mechanisms for lightweight detection and recovery of errors, and 3) an analysis of algorithm performance that quantifies resilience in presence of errors. Our results show how algorithm resiliency can already be provided for "free" (i.e. without hardware support), and how we can extend operation with additional lightweight mechanisms. These results reinforce the need to study algorithms to direct and/or inspire architecture approaches.

This paper is organized as follows: in Section II, we present examples of now algorithm resilience is impacted by computational noise. In Section III, we define a set of lightweight architectural mechanisms to allow the algorithms to operate in different levels of injected errors. In Section IV, we describe our simulation setup for selected set of algorithms, including a methodology to inject computation noise. In Section V, we provide a detailed set of analysis that evaluates the resilience of the architecture and the performance offered by the proposed lightweight mechanisms. Finally in Section VI, we present our conclusions and discuss our future work in this space.

II. Analyzing Computational Noise

In this section, we show the statistical nature of an error distribution (i.e. shape of the noise) to illustrate how algorithms are impacted. In Figure 1, we show a statistically generated distribution of image pixel values due to bit flip errors for any bit in 8-bit pixel. That is, we generate this graph over millions of pixels in an image with each bit in a pixel having a probability of corruption. We show that the distribution of errors to the original image, for each pixel, has a Gaussian shape for three different cases. The shape is data dependent, and can be considered as the convolution of the individual bit error rate (shown in the Figure 1 inset) against the data values.

For the "No Correction" case, the distribution of errors is flat with equal spread, representing random output over the data range. This means that the algorithm is faced with low signal-to-noise ratio because the entropy of data is at a point where results are no longer useful. For the "1MSB" case, we show a distribution of errors when the most significant bit is corrected. The noise shape shows that we can expect our algorithms to be more resilient because most errors are within a smaller range of values centered on zero. For the "4MSB" case, we show a very narrow distribution of error where there is only a small deviation from zero-mean for high error resilience. Using this approach, we can understand how computational noise can modify the entropy of data to a point where the results are no longer useful. It is this understanding of data entropy that can offer insights into opportunities to increase power efficiency with computational noise.

It is straightforward to understand that the more bit errors corrected, the better the performance of the algorithm. It is also easy to understand that the MSB has more value significance than the LSB (least-significant-bit). It is important to note, however, that from an error distribution perspective, by correcting the MSBs, we are preserving the mean-center distribution. That is, as the entropy of data is greater (i.e. flatter distribution), we can remove the tail end of the distribution by correcting the MSB bits. Here, we use bit error correction to illustrate a point on noise distribution, and there are other mechanisms to correct the bit errors.

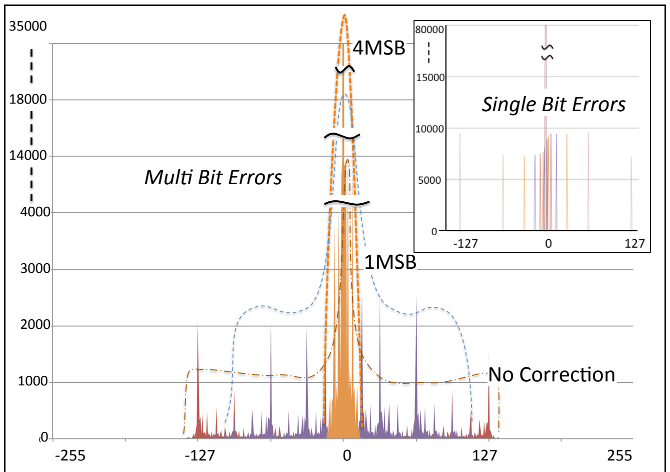

Figure 1. Statistical distribution of pixel values due to injected bit errors. Mean centered around zero indicates approximate data value range as well as algorithmic resilience.

We describe a case study that evaluates the efficacy of the computational noise approach. We choose a video stabilization algorithm because it has some of the characteristics of neural algorithms, namely, iterative and converging. We simulated this algorithm in C++ reading in raw image files from memory. We then simulated soft errors for the internal data structure to emulate computational noise induced by memory. Figure 2 illustrates the effects of computation noise on the processed image. An original image is read to generate a Laplacian image [4], and then a motion vector is generated. We show in Figure 2 (right most) effects of computational noise at 10% probability of bit flip for each bit in a pixel.

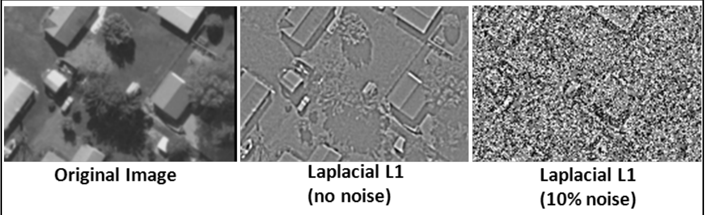

Figure 2. Level-1 Laplacian Pyramid [4] images showing the effects of computational noise (memory bit flips).

Figure 3 describes the algorithmic results for different levels of computational noise (represented by bit flip probability). The output of the global-motion estimation (GME) algorithm is a motion vector, representing camera movement. We calculate the error against a ground truth by calculating the Euclidean distance among the motion vectors, which is described as transformed position of the image corner and the center point. Zero pixel error would be ideal but single to low digit pixel errors are tolerable, which is represented by slight camera jitter with only a few pixel shift.

The results show a range of algorithmic operation based on computational noise. Certainly, without any error correction, the algorithm would not tolerate beyond 5% noise. However, there is still a range of safe-operational conditions since there are negligible levels of errors. With simple error corrections for the top-most or top four MSB, the algorithm can operate in higher levels of computational noise.

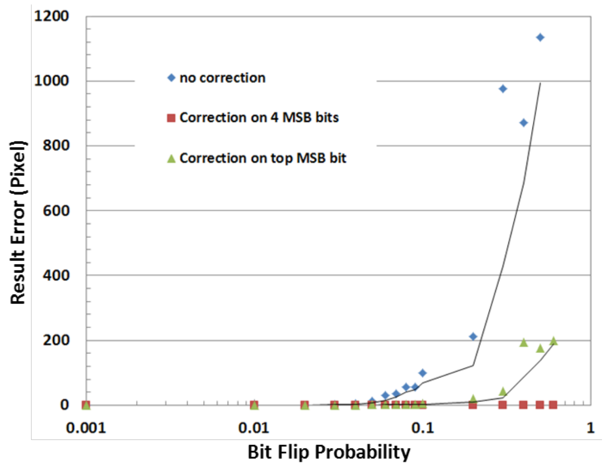

Figure 3. Noise resilience of video stabilization algorithm under different levels of computational noise and lightweight recovery mechanisms (e.g. MSB corrections)

To further enforce the notion of algorithmic resiliency, we provide evidence with respect to the key sources of data redundancy that enables robustness against noise. As others have noted, recognition, mining, synthesis (RMS) applications are the emerging important probabilistic applications [5,6], with opportunities for optimization. For this paper, we would describe data redundancy in the context of computer vision applications for object recognition.

Spatial redundancy. Spatial redundancy refers to the well-known property that an image pixel has similar value to its neighboring pixels. This property is already used by some image compression technique such as JPEG. Spatial redundancy provides error resilience for image processing algorithms since the error occurred for processing one pixel can be corrected based on values from neighboring pixels.

Temporal redundancy. Similar to spatial redundancy, where redundancy exists in a single image, the adjacent image frames in a video also carry similar pixel values [7,8]. Temporal redundancy also provides error resilience for certain computer vision algorithms. For example, in the human tracking surveillance applications, errors occurred in the processing of one frame can be disregarded, and the processing of following sequences can still track the human.

Kinematic redundancy. Kinematic redundancy comes from an understanding of the physical object being detected and tracked. An object model with physical attributes such as speed and behavior can be used to remove false alarms from spurious noise and incorrect inferences from other algorithmic elements. For example, because we understand the maximum speed of a human on foot, we can remove detected objects as non-humans if they exceed the speed criterion. In examples for gesture recognition, the connectivity between our body parts using human skeleton modeling can be used to distinguish and narrow down body poses based on what is physically possible.

III. Architectural Mechanisms

In this section, we describe the hardware mechanisms to support lightweight detection and recovery. We begin by noting existing research in approximate computing and voltage scaling. For example, there is prior work [5,9] that proposes to approximate original code region using neural processing unit for better performance and energy efficiency. Other researchers have proposed to build the logic elements with lower precision or accuracy [10]. There is also a body of research involving lowering of the operating voltage. Prior work [11] proposed a dual-voltage design where components can run in a high voltage that supports a precise and reliable operation. Some microarchitectural components such as register file, functional units and data caches can run in the lower voltage for better energy efficiency. The effect of running at a lower voltage is the reduced reliability of the hardware: the read and write accesses to SRAM array might fail and the functional units might produce the wrong values.

For brevity, we describe our design as having a simple error check and value speculation (or prediction) to recover from errors. Figure 4 shows a high level diagram of a Data Speculation Logic module, comprising of a simple error check (e.g. bit parity) and a buffer. The history value buffer stores the operands for the approximate instructions only. Similarly to register renaming, the entry in the buffer for approximate instruction can be allocated in the decoding stage. Once the error is detected, instead of supplying the faulty value, the value stored in the history buffer is supplied. If no error is detected, the corresponding entry in the history buffer is updated with the correct value. Depending on whether we want to rely on spatial, temporal, or kinematic redundancy, we can create a set of heuristic mapping for the history buffer. We can also simply turn off the data speculation logic and let the error peculate through.

As we've noted in the previous sections, noise is acceptable part of the computation. We describe the use of a history buffer, but other value prediction logic can be used instead. The intent of this study is to show a mechanism where the amount of noise and the data value can be controlled.

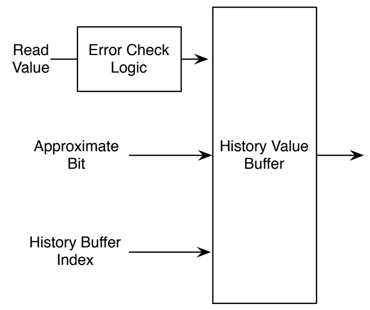

Figure 4. Generic diagram of a Data Speculation Logic module

From a higher level architecture perspective, we anticipate a example configuration that adopts the dual-voltage design, as shown in Figure 5. Components filled with white color, such as instruction fetch & decode, out of order execution engine, functional units and load store queue, can only run in the high voltage mode, i.e. run in precise mode. In contrast to that, components filled in yellow color can run both high voltage (precise mode) and lower voltage (approximate mode). Blue components are special component that we propose to include the data speculation logic. Although the design is very similar to dual-voltage approach [11], we note the main difference in using data speculation for improvements based on our understanding of data redundancy. This allows the miroarchitectural elements such as register files to run exclusively in lower voltage mode.

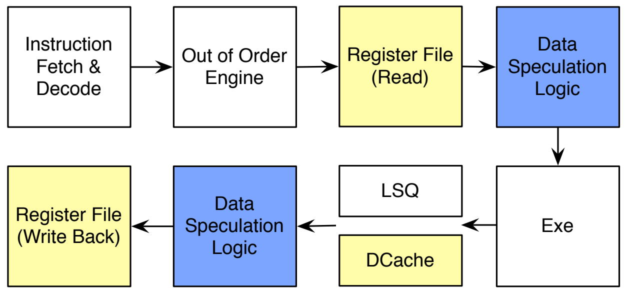

Figure 5. Generic architectural diagram using the Data Speculation Logic module

IV. Experimental Setup

In our initial evaluation for the Data Speculation Logic module, we consider several orthogonal design decisions. First, we consider the mapping for the history buffer, i.e. which data value to copy. In this paper, we show results where we (a) copy value from the adjacent pixel location, and (b) copy from the previous instance of the same instruction. In exploring these mapping choices, we are hoping to find the impact of noise resilience based on speculating data from spatial redundancy. Because the software is often coded where image data is accessed in raster-scan, spatial locality is often maintained within a window of data access.

Second, we consider the level of "data protection". That is, we can (a) protect the entire word such that the buffer will always provide data without any noise, and (b) partial protection of the high order MSB whereby the LSB is allowed to contain noise. We choose to explore this aspect in order to expand on the notion of shaping the statistical distribution of injected noise, as shown previously in Figure 1.

In this section, we describe our simulation methodology using an error injection mechanism directly during program execution. We describe the algorithms under evaluation and the associated metrics. Then we show our evaluation of the proposed architectural mechanisms.

A. *Error Injection*

Our error injection framework is based on Pin [12] as shown in Figure 6. Since Pin shares the same virtual memory address space with the instrumented program, the program can directly specify which region of memory can be injected with errors.

There are two phases of execution for better performance. The first phase consists of a single run to collect information for instructions to inject errors. Based on the profiling results, the second phase consists of multiple run of the instrument program for statistical data collection. As such, the data collect time is reduced considerably.

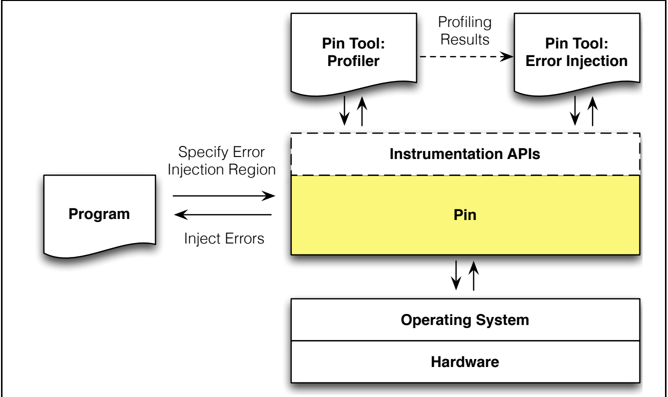

Figure 6. Pin Based Error Injection Framework Overview

The data error is only injected for instructions defined in nested loops. In these regions, matrix based access to data is common, e.g. row and column access to pixel data in image processing and computer vision algorithms. We anticipate that these instructions are destined for "approximate mode", we inject memory read errors using Pin in this area. Please see Figure 7 for an example code snippet.

It is important to note that the Pin error-injection method is used here to facilitate simulation only. There is certainly a line of research and development available towards automatic detection of code snippets for NTV operation, and other methods to select the MSB and LSB of bit lengths in different data types. Although right now we manually label instructions as approximate, prior work [3,9] on high level programming language for approximate computing would also suit our framework.

```
for ( unsigned i = 0; i < height, i ++ ) {
  for ( unsigned j = 0; j < width, j ++ ) {
    // Approximate region: start noise injection
  }
}
```
Figure 7. Example codes snippet for error injection

B. *Evaluated Algorithms*

We focus on video stabilization application, which consists of multiple algorithmic components. For each access to memory buffers in these algorithmic elements, we inject data error based on a set probability of error per bit.

Pyramid. An image pyramid is a type of multi-scale signal representation of the original image. It consists of a sequence of copies of an original image in which both sample density and resolution are decreased in regular steps, as shown in Figure 7. Each level of the pyramid images consists a Gaussian and Laplace image. The bottom, or zero level of the pyramid, G0, is equal to the original image. This is low-pass filtered and subsampled by a factor of two to obtain the next pyramid level, G1. G1 is then filtered in the same way and subsampled to obtain G2. Further repetitions of the filter subsample steps generate the remaining pyramid levels. Pyramid [4,13] is very important in computer vision area since it provides a common framework for implementing highly efficient analysis algorithms as well as an architecture for special purpose image processing hardware.

Global Motion Estimation. Global motion estimation (GME) is widely used for video stabilization [14] and compression [15]. GME benchmark implements the hierarchical algorithm described in [16], as shown in Figure 8. The motion estimation is performed on difference levels of pyramid images instead of the source image level. Hierarchical approach adopts a coarse-fine refinement strategy, i.e. the estimation is performed at the lower resolution pyramid images first then higher resolution levels for better accuracy. Thus the pyramid-based approach is more computation efficient than algorithms processing the original image.

Output Quality Metric. For each benchmark, we have a metric for deciding the quality of its result. For Pyramid, the signal to noise ratio (SNR) is used. For GME, to derive the quality metric, we first apply the estimated motion vector to selected five points (four corners and one center). Then we apply the ground truth motion vector to the same points. The result quality is derived by calculating the Euclidean distance among transformed position of these points with estimated motion vector and ground truth motion vector.

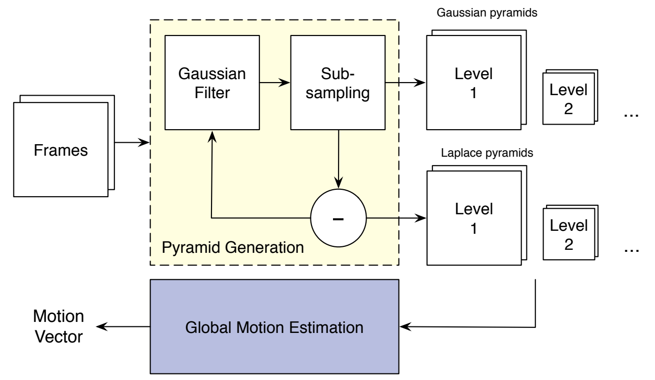

Figure 8. Algorithmic Overview of Video Stabilization

V. Results and Evaluation

We first present the evaluation of the algorithm resilience of selected algorithms described earlier. We analyze the error resilience property by inspecting how the output result quality changes with different bit flip probability when running in the approximate mode. We consider the four different design options to map the buffers in the Data Speculation Logic module.

In Figure 9, we show the simulation results for different bit flip probability against output quality metric. In general, result errors in the order of a single pixel are tolerable. We show three baseline comparisons with the first two graphs. The "Baseline" represents computational noise injected without any checking or recovery. The "Error in source image" represents sensor-oriented noise at the input without any computational noise, representing the traditional computing without NTC (and obviously, without the power benefits of NTC). The "ECC-Single Error Detection" represents traditional ECC applied as error recovery.

We evaluate two versions of benchmark GME: using only level 1 of the pyramid images (Figure 9a) and using multiple levels (Figure 9b). From the two plots, we see that single level GME's output is less resilient compared to the GME using multiple levels, which we anticipated based on spatial redundancy. We see and anticipated that, in comparison with base line, computational noise is more demanding than just sensor noise (our base line data). That is, without NTC and higher power computation, the algorithm can tolerate sensor noise up to these levels. For multi-level GME, we also injected errors in every level of the pyramid, so the intrinsic robustness of the algorithm can maintain resiliency from input sensor noise up to p=0.04 (4% bit flip probability).

Based on our results, we find that there is negligible difference in performance for cases where the Data Speculation Logic module gathers data from adjacent memory location or from data used previous instruction. This confirms our anticipation that the image access pattern is raster-scan and spatial locality is therefore maintained within a window of data access. From this, we can also conclude that the Data Speculation Logic module can be simplified because the history buffer is tied closely with instruction stream rather than memory access. More specifically, we are suggesting that the logic to maintain the mapping in the history buffer is greatly simplified when we are just considering data from most recently used instructions.

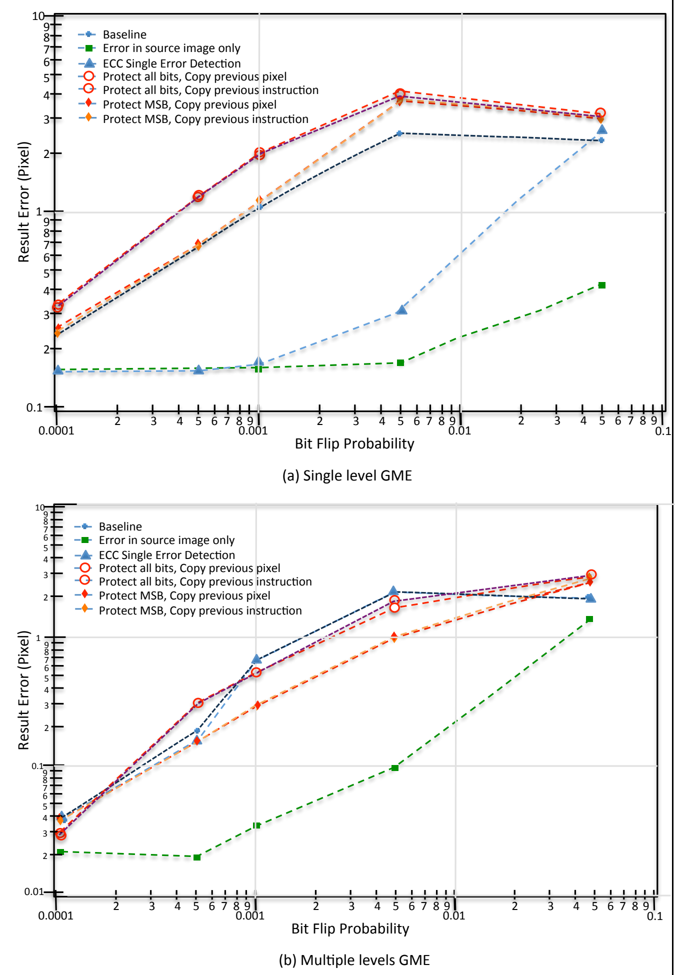

Figure 9. Evaluated resiliency under different data speculation schemes

With respect of the design choice related to level of protection for the bits, we find that it is fine to leave the LSB noisy. In fact, the results show that if we copy the duplicate the entire pixel data, the system has a higher overall error. Algorithmically, we note that copying the previously used pixel may be fast and simple, but it will introduce horizontal smearing due to data duplication. This smearing affects the motion vector with duplicated data. In comparison, with noisy LSB (e.g. protect and copy only MSB from previous instruction), we leave the algorithm with statistical noise in the motion estimation algorithm, without throwing off the convergence. More specifically, the smearing effect from data duplication is more likely to indicate camera motion, while pixel noise (from LSB) would not. From this result, we can infer that the history buffer can be greatly simplified because we just need to store the upper MSB, while LSB can be noisy.

VI. Conclusions and Future Work

In this paper, we explore the notion of computation noise and its statistical nature to arrive at an alternative approach for resiliency. We presented a very early design of a lightweight error checking and recovery mechanism to extend noise resiliency beyond the traditional "safe operating region" of the algorithm. We describe a microarchitecture design with results of how approximate computing can be extended with data value speculation. We show that with an understanding of the spatial, temporal, and kinematic data redundancy, we can arrive at a greatly simplified design, while maintaining algorithmic performance.

Going forward, there is much more we can do as future work to extend the idea of noise as an architecture and circuit design parameter for data value approximation. Specifically, we would like to merge the approach with approximate computing concepts such as lower precision and loop perforation. With the computational noise approach, we would instead provide full precision data, but in presence of noise.

Our intent is focused on showing what are the algorithmic performances given some controllable source of computational noise. We point to the idea that lightweight mechanisms could be afforded to maintain algorithmic performance. While our paper describes only a small fraction of the processor microarchitecture can operate in NTV, we expect to explore and expand into larger proportion of the chip. Our initial design using a history value buffers are offered as evaluation of possible architectural mechanisms, but other, smaller logic circuit could be used instead for data speculation. We are not at a point where we can generalize a circuit for data speculation.

Furthermore, there are number of VLSI design rules that can be reconsidered because we are now able to leverage noise in the design parameter. For example, we can save power by lowering voltage and introduce crosstalk in interconnects, for example. We may even have non-symmetrically interconnect widths for a bus (e.g. purposefully lay out a network where the least significant bits – LSB – are narrower and thus noisy, to improve wiring density.

REFERENCES

[1] Ronald G. Dreslinski, et al. "Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits." Proceedings of the IEEE 98.2 (2010): 253-266.

[2] Smruti R. Sarangi, et al. "VARIUS: A model of process variation and resulting timing errors for microarchitects." Semiconductor Manufacturing, IEEE Transactions on 21.1 (2008): 3-13.

[3] M. Corbin, et. al., “Proving Acceptability Properties of Relaxed Approximate Programs”, PLDI’12.

[4] C.H. Anderson, J.R. Bergen, P.J. Burt ,and J.M .Ogden, “Pyramid methods in image processing,” 1984.

[5] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger,“ Neural Acceleration for General-Purpose Approximate Programs,” in Proceedings of International Symposium on Microarchitecture (MICRO), 2012.

[6] L. Leem, H. Cho, J. Bau, Q. Jacobson, and S. Mitra, “ERSA: Error Resilient System Architecture for probabilistic applications,” in Proceedings of Design, Automation Test in Europe Conference Exhibition (DATE), 2010.

[7] H. Lu, B. Wang, X. Xue, and Y.P. Tan, “Effective Shot Boundary Classification Using Video Spatial-Temporal Information,” in Proceedings of International Symposium on Circuits and Systems (ISCAS), 2005.

[8] L.-Q. Xu and Y. Li, “Video Classification Using Spatial-temporal Features and PCA,” in Proceedings of International Conference on Multimedia and Expo (ICME), 2003.

[9] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger., “Architecture Support for Disciplined Approximate Programming,” in Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2012.

[10] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, “IMPACT: IMPrecise Adders for Low-power Approximate Computing,” in Proceedings of International Symposium on Low Power Electronics and Design (ISLPED), 2011.

[11] D. Mohapatra, V. Chippa, A. Raghunathan, and K. Roy, “Design of voltage-scalable meta-functions for approximate computing,” in Proceedings of Design, Automation Test in Europe Conference Exhibition (DATE), 2011.

[12] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S.Wallace, V. J. Reddi, and K. Hazelwood, “Pin: building customized pro- gram analysis tools with dynamic instrumentation,” in Proceedings of Conference on Programming Language Design and Implementation (PLDI), 2005.

[13] P. Burt, “A Pyramid Framework for Real-Time Computer Vision,”in Foundations of Image Understanding, ser. The Springer International Series in Engineering and Computer Science. Springer US, 2001.

[14] T.-S. Wang, S.-J. Kang, K.-Y. Byun, T.-C. Kim, and S.-J.Ko, “Robust global motion estimation for video stabilization,” in Global Conference on Consumer Electronics (GCCE), 2012

[15] Y. Keller and A. Averbuch, “Fast gradient methods based on global motion estimation for video compression,” Circuits and Systems for Video Technology, IEEE Transactions on, 2003.

[16] J. Bergen, P. Anandan, K. Hanna, and R. Hingorani, “Hierarchical model-based motion estimation,” in Computer Vision ECCV’92, ser. Lecture Notes in Computer Science, 1992, vol. 588.