A Low Overhead Error Confinement Method based on Application Statistical Characteristics

Interactive presentation

***Abstract*—Reliability has emerged as a critical design constraint especially in memories. Designers are going to great lengths to guarantee fault free operation of the underlying silicon by adopting redundancy-based techniques, which essentially try to detect and correct every single error. However, such techniques come at a cost of large area, power and performance overheads which making many researchers to doubt their efficiency especially for error resilient systems where 100% accuracy is not always required. In this paper, we present an alternative method focusing on the confinement of the resulting output error induced by any reliability issues. By focusing on memory faults, rather than correcting every single error the proposed method exploits the statistical characteristics of any target application and replaces any erroneous data with the best available estimate of that data. To realize the proposed method a RISC processor is augmented with custom instructions and special-purpose functional units. We apply the method on the proposed enhanced processor by studying the statistical characteristics of the various algorithms involved in a popular multimedia application. Our experimental results show that in contrast to state-of-the-art fault tolerance approaches, we are able to reduce runtime and area overhead by 71.3% and 83.3% respectively.**

I. Introduction

The aggressive shrinking of transistors have made circuits and especially memory cells more prone to parametric variations and soft errors that are expected to double for every technology generation [1], thus threatening their correct functionality. The increasing demand for larger on-chip memory capacity, predicted to exceed 70% of the die area in multiprocessors by 2017 is expected to further worsen the failure rates [2], thus indicating the need for immediate adoption of effective fault tolerant techniques.

Techniques such as Error Correcting Codes (ECC) [3] and Checkpointing [4] may have helped in correcting memory failures, however they incur large area, performance and power overheads ending up wasting resources and contracting with the high memory density requirements. In an effort to limit such overheads recent approaches exploit the tolerance to faults/approximations of many applications [5] and relax the requirement of 100% correctness. The main idea of such methods is the restricted use of robust but power hungry bit-cells and methods such as ECC to protect only the bits that play a more significant role in shaping the output quality [6] [7]. Few very recent approaches exist also that extend generic instruction sets with approximation features and specialized hardware units [8][9][10]. Although such techniques are very interesting and showcased the available possibilities in certain applications they are still based on redundancy and have neglected to exploit some more fundamental characteristics of the application data.

Contribution In this paper, we enhance the state of the art by proposing an alternative system level method for mitigating memory failures and presenting the necessary software and hardware features for realizing it within a RISC processor. The proposed approach, instead of adding circuit level redundancy to correct memory errors tries to limit the impact of those errors in the output quality by replacing any erroneous data with the best available estimate of those data. The proposed approach is realized by enhancing a common programming model and a RISC processor with custom instructions and low cost hardware support modules. We demonstrate the low overhead and error mitigation ability of the proposed approach by applying it on the different algorithmic stages of JPEG and comparing with the extensively used Single Error Correction Double Error Detection (SECDED) method. Overall, the proposed scheme offers better error confinement since it is based on application specific statistical characteristics, while allowing to mitigate single and multiple bit errors with substantially less overheads.

The rest of work is organized as following. Section II introduces the proposed approach while Section III describes the enhancements of a processor for realizing it. Section IV presents the statistical analysis of the proposed approach. Section V presents the simulation results. Finally, Section VI concludes the work.

II. Proposed Error Confinement Method

Assume that a set of data $d \in \mathcal{D} = \{d_1, \ldots, d_K\}$ being produced by an application are distributed according to the probability mass function $P_d(d_k) = \Pr(d = d_k)$. Such data are being stored in a memory, which is affected by parametric variations causing errors (i.e. bit flips) in some of the bit-cells. Sure errors eventually result in erroneous data leading to a new data distribution $\bar{P}_{d_k}$. The impact of such faults can be quantified by using a relevant error cost metric which in many cases is the mean square error (MSE) defined as

$$\mathcal{C}(\bar{d}) \triangleq \mathbb{E}\big\{(d - \bar{d})^2\big\} \tag{1}$$

with the expectation taken over the memory input d. Our proposed method focuses on minimizing the MSE between the original stored data d and the erroneous data $\bar{d}$ in case of a-priori information about the error $\mathcal{F}$ through an error-mitigation function $d^* = g(\mathcal{F})$ which can be obtained by solving the following optimization problem:

$$d^* = g(\mathcal{F}) \triangleq \arg\min_{\bar{d}} \mathcal{C}(\bar{d} \,|\, \mathcal{F}). \tag{2}$$

where,

$$\mathcal{C}(\bar{d} \,|\, \mathcal{F}) \triangleq \mathbb{E}\big\{(d - \bar{d})^2 \,|\, \mathcal{F}\big\} \tag{3}$$

Basic arithmetic manipulations show that the resulting correction function is given by $g_{\text{MMSE}} = \mathbb{E}\{d[n] \,|\mathcal{F}\}$. This essentially corresponds to the expected value of the original fault-free data. Such expected values can be eventually determined offline through Monte-Carlo simulations or analytically in case that the reference data distribution is known already as in many DSP applications. Note that the above function depends on the applied cost metric that is relevant for the target application and other functions may exist that can be found by following the above procedure. In our paper, we focus on MSE which is relevant for many applications and especially for our case study that we discuss later.

III. Realizing the Proposed Error Confinement in a RISC Processor

The proposed Error-Confinement function requires a scheme for detecting a memory error for providing the needed a-priori information $\mathcal{F}$ and a look up table for storing the expected reference values to be used for replacing the erroneous data. Obviously the realization of such a scheme in a processor require i) the introduction of custom instructions and ii) micro-architectural enhancements that we discuss next.

We base the proposed enhancements on the RISC processor core IP from Synopsys Processor Designer, which consists of five pipeline stages as depicted in Figure 1, supports mixed 16/32 bits instructions, while the HDL implementation of the core is fully synthesizable. Note that for the detection of an error required in our scheme we propose the use of a single parity bit within each word which is sufficient for detecting a single error. By doing so we essentially limit the required overhead as opposed to ECC methods that require the addition of several parity bits for the detection and correction of a single or more errors.

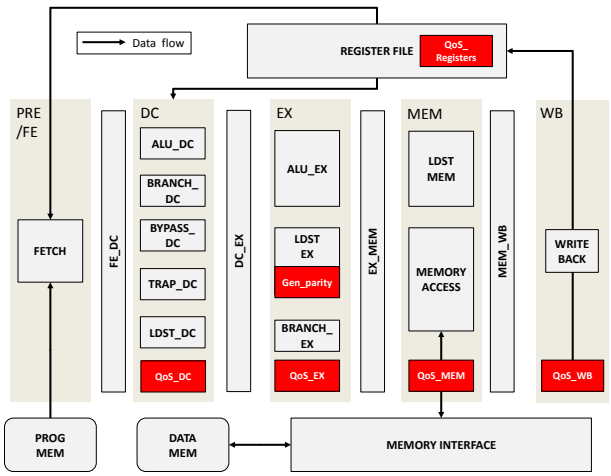

Fig. 1. Microarchitecture of RISC processor with enhancements for statistical based error confinement

A. Custom Instructions

At the assembly level we introduce 4 new instructions, which can be used either in standalone assembly or be embedded as inline assembly in a high-level language such as C/C++. To begin with we need to specify the start address and the word size of the memory block that is going to be protected by the proposed scheme and indicate the place in the look up table (LUT) as well its size, where the expected value to be used in case of an error is stored. To this end we introduce the following instruction: ***set_data*** @{***data_start***} @{***data_size***} @{***lut_start***} @{***lut_size***}
in which all arguments are provided using general purpose registers.

Furthermore, the instruction ***chk_load*** @{***dst***} @{***src***} @{***index***} is introduced for statistically confining the error in specific memory blocks while performing memory reads . In particular, before reading the protected data, this instruction detects any error within the read data in the register @{src} and in case i) of an error it replaces the erroneous data with the reference expected value stored in the position @{index} of the LUT and loads the value into the register @{dst}, while ii) in case of no error the register @{dst} is assigned directly to the correct value kept in the register @{src}.

Finally, to enable the protection of specific memory write accesses we introduce the instruction ***en_parity*** as well as the instruction ***dis_parity*** for disabling the protection of any data if needed. The above instructions are incorporated in the newly constructed LLVM based C compiler (through the use of Synopsys Processor Designer), which supports instruction set extensions using inline assembly.

B. Micro-Architectural Enhancements

The introduced instructions require the enhancement of the microarchitecture of the target RISC processor with customized modules which are highlighted in Figure 1. The detailed functionality of the logic functions within each module in each pipeline stage is described in detail in Figure 2.

IV. Case Study and Statistical Analysis

A. Case Study - JPEG

To demonstrate the efficacy of the proposed scheme we use as case study the JPEG, which is a widely used lossy compression technique of digital images that became a popular application example among error resilient techniques. JPEG consists of several stages including color space transformation and down sampling. In this work, we focus on the subsystem shown in Figure 3 which consists of four major procedures. In particular an input image of size 512×512 is decomposed into 4,096 matrices of the size 8×8. Then each matrix is being processed individually by the 2D Discrete Cosine Transformation (2-D DCT) [11] that essentially transforms the image into the frequency domain producing the

QoS_DC — DC stage module to decode 4 new instructions in ISA:
1. ***en_parity_st***: Enable parity encoding
2. ***dis_parity_st***: Disable parity encoding
3. ***set_data***: Register protected data range and reference LUT
4. ***chk_ld***: Check the data using parity value

Gen_ parity — EX stage module, implement following operation:
1. ***parity*** in MEM stage is activated in case the stored data lies in the protected range

QoS_EX — EX stage module, implement following operation:
1. ***chk_ld_ex***: Activated by ***chk_ld***, verify parity bit of the incoming data. In case of missmatch, activate ***chk_ld_mem*** in MEM, otherwise activate ***chk_ld_wb_correct***

QoS_ MEM — MEM stage module, implement following operations:
1. ***parity***: Main parity encoding logic. Each 32 bits data generates 1 bit parity. Encode 32 words as 1 word parity, activate ***partiy_st*** logic in WB stage 32 bits parity are ready
2. ***chk_ld_mem***: When a parity missmatch is detected, get the address of corresponding LUT element and issue memory read. Activate ***chk_ld_wb_wrong*** in WB stage

QoS_WB — WB stage module, implement following operations:
1. ***parity_st***: Store 32 bits parity as one data word into data memory
2. ***Chk_ld_wb_wrong***: In case of parity missmatch, return the reference value from LUT
3. ***Chk_ld_wb_correct***: No parity missmatch, return the input value from instruction

QoS_ Registers — Special registers in RegisterFile,
1. set and re-set the processor state for load check.
2. Record the location for LUT and runtime offset to access the reference data.

Fig. 2. Introduced modules and their functionality

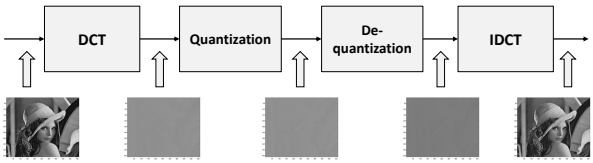

Fig. 3. Subsystem in JPEG application

DCT coefficients as output which are then finally being quantized. For the reconstruction of the image De-quantization and 2D Inverse Discrete Cosine Transformation (2-D IDCT) are applied. In general, the quality of the output image compared to the original one is evaluated using the peak signal to noise ration (PSNR) [12] and a typical PSNR value for a lossy image is 30 dB.

B. Statistical Analysis of JPEG

Following the steps of the proposed approach we statistically analyzed the different stages of JPEG by performing several simulations with different images. Simulations show that the output matrices of DCT and quantization share a similar pattern; the elements at the top-left corner of both DCT and quantization output matrix are larger in magnitude compared to the rest which in most cases are close to zero. Figure 4 shows the expected value of each element in the DCT and quantization output matrix after averaging their values across 4,096 individual matrices for over 10 images. Such values are used as the reference expected values for replacing the erroneous data in case of a detected memory error in our approach. Note that these values are stored in a LUT that was described in Section III.

V. Results

A. Experimental Setup

We have modified the RISC processor as discussed in Section II and enabled the injection of bit flips in the memory locations

$$\begin{bmatrix}
-242.538422 & -1.565797 & 0.646120 & 1.169245 & -0.020233 & 0.940231 & 0.027707 & 0.733925 \\
-0.222834 & -0.108308 & -0.780093 & 0.133814 & 0.039231 & 0.062401 & 0.025521 & -0.004177 \\
-0.649843 & -0.512487 & 0.228183 & 0.353566 & 0.000773 & 0.069678 & 0.012114 & -0.001641 \\
-0.641493 & 0.030510 & 0.412436 & 0.099906 & 0.021144 & -0.039063 & -0.002198 & 0.004568 \\
0.017609 & -0.038111 & -0.065816 & -0.025756 & -0.081390 & 0.005183 & -0.007455 & -0.005406 \\
-0.430538 & -0.021713 & 0.046092 & 0.007092 & -0.002983 & -0.000642 & -0.005067 & 0.005527 \\
0.084360 & 0.018600 & -0.003267 & -0.000333 & -0.004912 & 0.004006 & 0.001126 & -0.009209 \\
-2.868614 & 0.005601 & -0.002588 & 0.005061 & 0.000778 & -0.002361 & -0.002779 & -0.003040
\end{bmatrix}$$

(a) Average 8x8 matrix after DCT operation

$$\begin{bmatrix}
-15.150879 & -0.142334 & 0.065918 & 0.074463 & -0.002197 & 0.003174 & 0.000000 & 0.000000 \\
-0.015137 & -0.008301 & -0.055176 & 0.002930 & 0.000000 & 0.000244 & 0.000000 & 0.000000 \\
-0.045898 & -0.039795 & 0.015381 & 0.003174 & -0.001953 & 0.000000 & 0.000000 & 0.000000 \\
-0.035645 & 0.001953 & 0.018311 & 0.002441 & 0.000244 & 0.000000 & 0.000000 & 0.000000 \\
-0.002197 & -0.002197 & -0.000488 & -0.000244 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\
-0.002686 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\
0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\
0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000
\end{bmatrix}$$

(b) Average 8x8 matrix after Quantization

Fig. 4. Reference matrix for DCT and Quantization Coefficients

```
int imageEx[SIZE][SIZE];
int imageData[ROW_SIZE][COL_SIZE];
int imageExtended[ROW_SIZE][COL_SIZE];
// reference LUT containing generalized data
int lut_imageEx[8][8]={{780, -1, 0, 1, 0, 0, 0, 0 },
                       {  0,  0, 0, 0, 0, 0, 0, 0 },
                       {  0,  0, 0, 0, 0, 0, 0, 0 },
                       {  0,  0, 0, 0, 0, 0, 0, 0 },
                       {  0,  0, 0, 0, 0, 0, 0, 0 },
                       {  0,  0, 0, 0, 0, 0, 0, 0 },
                       { -2,  0, 0, 0, 0, 0, 0, 0 }};
int DiscreteCosine(int imageData[SIZE][SIZE], int imageEx[SIZE][SIZE])
{   ......
    asm("enable_parity");      // turn on store protection
    imageEx[i1][j1]= (int)sum;
    asm("disable_parity");     // turn off store protection
    .......
}
int Quantization(int imageEx[SIZE][SIZE], int imageExtended[SIZE][SIZE])
{   ......
    int src = imageEx[i1][j1];
    int dst;
    asm("check_value @dst, @src, (8*i+j)");  // automatic correction
    imageExtended[i1][j1]= (dst/quant[i][j]);
    ......
}
int main()
{   ......
    // register range of protected data using reference LUT
    asm("set_data @imageEx, 262144, @lut_imageEx, 64");
    DiscreteCosine(imageData, imageEx);
    Quantization(imageEx, imageExtended);
    ......
}
```

Fig. 5. Programming example with custom instructions for DCT

storing the images and intermediate results of the JPEG. Note that no errors are injected on instruction cache and other registers which are assumed to be adequately protected.

For detecting errors as we said we encoded each of the 32-bit data of the application with a single parity bit which is sufficient for detecting a single fault. Following the proposed method the new instructions were used as inline assembly to describe JPEG as shown in Figure 5. In this example an array containing the reference expected values for the DCT coefficients is defined. Within the DCT function, before performing a store to the memory, parity encoding is enabled, which is turned off after a write-store operation. Within the quantization function, the load check is performed whenever a value is read out from the array where the DCT coefficients are stored for replacing it with the relevant expected value in case of an error.

The above code was compiled and executed on the modified processor and the performance, power and quality were measured under different error rates as discussed next. Note that for comparison we replicated a similar infrastructure by using a conventional SECDED Hamming code scheme $H[38, 32]$ for the protection of the specific memories (protected by our scheme), which requires 6 parity bits for encoding each 32 bit memory word.

B. Evaluation of Quality

Figure 6 shows the output images and corresponding PSNR values with different numbers of injected bitflips according to typical error rates in 65nm process technology. The results show that in case of 800 and 1000 bitfilps, the output image is degraded by 7.6% and 41.2% compared to the error free case.

The reason for such a large degradation in case of 1000 bitflips is that two bitflips in the same data word are allowed which cannot be detected by the single bit parity. Careful examination of our simulations indicated that some of such double bitflips affected words that relate to the first 20 DCT coefficients of the 8×8 matrix (remember there are 4092 such matrices in each image). As other works have also shown such coefficients control almost 85% of the overall image quality and thus if the get affected by errors and these are not tackled by any means as in this case then they lead to significant quality degradation.

As we mentioned we compared the quality achieved by our approach with a SECDED ECC. Figure 7 shows the obtained results in case of protecting the output DCT and quantization coefficients wth the two schemes under different number of single bitflips. We observed that as the number of the injected single bitflips increases, the output quality (interms of PSNR) achieved by using the proposed approach is slightly less than that achieved by using the ECC scheme. This can be attributed to the fact that in some cases the correct value of the erroneous data that is being

Fig. 6. Output images under different schemes of error injection

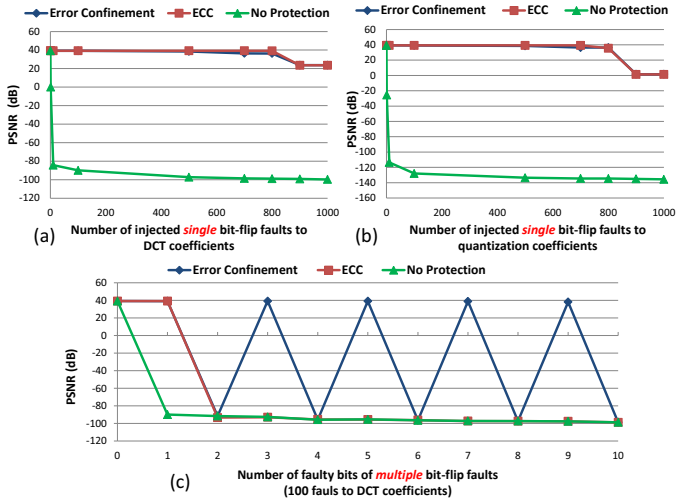

Fig. 7. PSNR under no protection, proposed scheme and ECC

substituted by the expected value may indeed lie in the tale of the distribution and thus may be far from the used reference expected value. In these cases the replacement will not be as accurate and thus the quality achieved by our approach may not be as perfect. In any case as we discussed our approach tries to confine the impact of memory errors by essentially approximating erroneous data with their expectation and sometimes such an approximation may not be as good. However, note that the proposed approach still achieves to provide output images with PSNR above 36 dB even under 800 bitflips, closely approximating the error free image.

We observed that above 800 bitflips (when double bitflips are allowed in each word) both methods fail to produce a good enough image, since neither scheme is able to detect and mitigate from multiple bitflips in single data word. In particular, on one side the SECDED ECC intrinsically cannot correct more than one error in a word and on the other side the single parity bit used to in the proposed scheme cannot detect two bitflips in a word and thus it does not engage the replacement of the erroneous data.

Our results reveal also a different aspect in the JPEG application. In particular, we observed that in case of more than 800 bitflips when double bitflips are taking place in each word then any untreated error in quantization coefficients are far more severe (causing large quality degradation) compared to untreated errors in DCT coefficients. This can be attributed to the sparse nature of the quantization coefficients (i.e. most of them are zero) and thefact that any untreated error will significantly alter the expected distribution of these data.

In addition to the above experiments, we have also evaluated the ability of our approach to address multiple bitflips in a single data word by replacing it with the expected reference value. Figure 7c) shows the achieved PSNR under different number of bitflips

in each word. We can observe that the proposed scheme helps to obtain a PSNR of more than 38dB (for the particular image) in case of odd number of faulty bitcells (when the parity bit can detect the error) while the PSNR degrades a lot in case of even number of faulty bit cells (which cannot be detected by a single parity bit). On the contrary, note that the SECDED ECC even with the use of 6 parity bits fail to address any number of multi bitlfips requiring more complex ECC schemes with much more parity bits. All in all, the proposed approach even with the use of single parity bit is able to address adequately the cases of odd multi flipbits in a single word. The addition of another parity could be employed to improve the capability of error detection which is left for future experimentation. The essential conclusion is that the replacement of erroneous data with an expected value suffices to confine the impact of single or even multi memory bitflips.

C. Performance and Power Results

The proposed enhanced processor is synthesized in 65nm Faraday technology and the power, performance and area results compared to the original processor are shown in Table I. Note that the reference processor in this case does not employ any protection scheme and our results in this paragraph try to reveal the overheads involved in enabling preferential protection of specific parts of a memory with special instructions as well as the cost of the proposed data replacement scheme. We can observe that the performance is decreased by only 4.2% but the instruction extensions for realization of the proposed scheme by a generic programming environment have resulted in large power and area overheads. The extra logic and registers for specifying the protected memory addresses (which is a unique and desirable feature in todays error resilient systems enabled by the proposed extensions), the added LUT and the 1-bit parity encoding are responsible for such overheads. However, note that implementing the same instruction extensions by using 6parity bits as needed by a H(38,32) ECC will result in much larger overheads.

TABLE I
RESULTS FOR THE PROPOSED ARCHITECTURE EXTENSIONS COMPARED TO THE REFERENCE UNPROTECTED PROCESSOR

	Area (NAND equiv.)		**Power (μWatt)**		**Critical path**
	Comb.	Seq.	Dynamic	Leakage	(ns)
Original	11789	6187	206	65	6.12
Proposed extensions	26519	10663	349	124	6.38
Increase (%)	124.9	72.3	69.4	90.8	4.2

To compare with the SECDED ECC we show the total time required for executing the JPEG application on a processor instance that involves our scheme and on another that implements the ECC. Figure 8 depicts the overall execution time of the JPEG application after processing images of different sizes from 8×8 till $1,024 \times 1,024$ and correcting randomly injected errors (in same locations) with ECC and the proposed scheme.

For small images both methods take similar time since the modules other than the ones shown in Figure 3 dominate the execution time. For images larger than 64×64 ECC takes significantly longer time compared to proposed scheme. In particular, for an image of size $1,024 \times 1,024$, ECC takes **3.5**$\times$ more time than the proposed scheme. Note that such overhead will further increase for larger images and more injected errors.

Although the architecture extension achieves large power overhead, the energy consumption ratio between proposed approach and ECC reduces as image size grows, which is illustrated in Figure 9. This is because ECC takes longer time to finish. Starting from image size of 128×128 the proposed approach consumes less energy than ECC, while the energy benefit increases even further for larger images.

Another interesting comparison to discuss is the difference in terms of memory usage. As we can also see in Figure 8 the proposed approach uses far less memory compared to SECDED ECC scheme which incurs 18.75% memory overhead in each protected data word. In particular, for an image of size $1,024 \times 1,024$, ECC

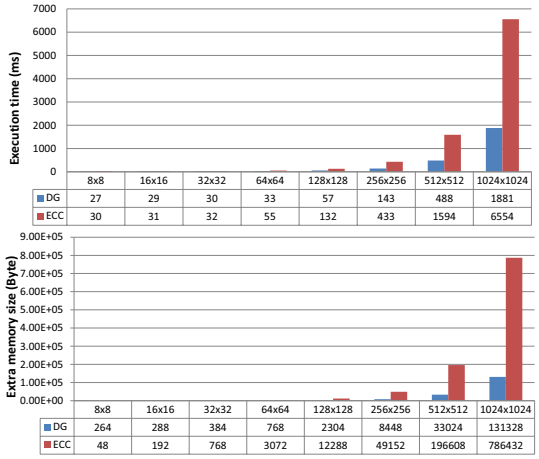

Fig. 8. Execution time and data memory usage under Error Confinement and ECC

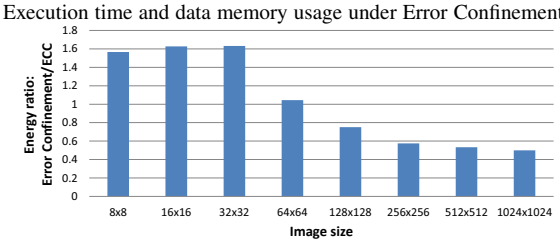

Fig. 9. Energy ratio between proposed approach and ECC vs image size

requires **5.99**$\times$ more memory than the proposed error confinement approach.

VI. CONCLUSION

In this work, a low cost error confinement technique is proposed which exploits the statistical characteristics of target applications and replaces any erroneous data with the best available estimate of that data. The architecture of a RISC processor with custom instructions supporting proposed approach is presented. The benchmarking result shows that the proposed approach achieves far less performance and memory usage overhead than ECC based error detection and correction, while also consumes less energy as image size grows. Further application-level studies using the proposed methodology will be presented in the future.

REFERENCES

[1] S. Borkar, "The exascale challenge," *Proc. VLSI-DAT*, 2010.
[2] T. Semiconductor, "Soft errors in electronic memory-a white paper," 2004.
[3] Y. Emre and C. Chakrabarti, "Techniques for compensating memory errors in JPEG2000," *IEEE Trans. VLSI Syst.*, vol. 21, no. 1, pp. 159–163, 2013.
[4] X. Dong, N. Muralimanohar, N. Jouppi, R. Kaufmann, and Y. Xie, "Leveraging 3d pcram technologies to reduce checkpoint overhead for future exascale systems," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ACM, 2009.
[5] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *Proceedings of the 50th Annual Design Automation Conference*, ACM, 2013.
[6] W. Yueh, M. Cho, and S. Mukhopadhyay, "Perceptual quality preserving SRAM architecture for color motion pictures," in *Design, Automation and Test in Europe, DATE 13, Grenoble, France, March 18-22, 2013*, pp. 103–108, 2013.
[7] I. Lee, J. Kwon, J. Park, and J. Park, "Priority based error correction code (ECC) for the embedded SRAM memories in H.264 system," *Signal Processing Systems*, vol. 73, no. 2, pp. 123–136, 2013.
[8] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," in *ACM SIGPLAN Notices*, vol. 47, pp. 301–312, ACM, 2012.
[9] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Quality programmable vector processors for approximate computing," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 1–12, ACM, 2013.
[10] 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 *ACM SIGPLAN Notices*, vol. 46, pp. 164–174, ACM, 2011.
[11] R. C. Gonzalez, *Digital image processing*. Pearson Education India, 2009.
[12] Q. Huynh-Thu and M. Ghanbari, "Scope of validity of psnr in image/video quality assessment," *Electronics letters*, vol. 44, no. 13, pp. 800–801, 2008.