Approximate Compressed Sensing for Hardware-Efficient Image Compression

Sai Praveen Kadiyala, Vikram Kumar Pudi and Siew-Kei Lam
School of Computer Science and Engineering, Nanyang Technological University, Singapore
Email: {saipraveen, pudi, assklam}@ntu.edu.sg

***Abstract*—Recently, compressive sensing has attracted a lot of research interest due to its potential for realizing lightweight image compression solutions. Approximate or inexact computing on the other hand has been successfully applied to lower the complexity of hardware architectures for applications where a certain amount of performance degradation is acceptable (e.g. lossy image compression). In our work, we present a novel method for compressive sensing using approximate computing paradigm, in order to realize a hardware-efficient image compression architecture. We adopt Gaussian Random matrix based compression in our work. Library based pruning is used to realize the approximate compression architecture. Further we present a multi-objective optimization method to fine tune our pruning and increase performance of architecture. When compared to the baseline architecture that uses regular multipliers on 65-nm CMOS technology, our proposed image compression architecture achieves 43% area and 54% power savings with minimal PSNR degradation.**

I. Introduction

Approximate computing has gained wide attention as it has been shown to lead to significant improvement in performance and power efficiency of computing systems [1]. Approximate computing provides area and power savings by trading off the accuracy of the results [2]. The reduced accuracy is acceptable in systems that take advantage of limited human perception capabilities. Such applications include multimedia (image and video processing), data-mining, search and weather forecast applications [3], [4], [5]. Approximate computing is also used to improve CPU performance. This is acheived by approximating the memory based computations and content addressable memory accelerators [6], [7]. Fig. 1 shows an example where energy savings are achieved at the expense of image output quality when FFT architectures with varying degrees of inexactness are employed [8].

Compressive Sensing (CS) is a recently developed technique that samples signals at sub-Nyquist rate and simultaneously compresses them using random projections [9]. Due to this, CS reduces communication cost and resources for compression. Generally, CS is applied to signals in the analog domain before they are converted to digital signals, thus minimizing power for signal compression [10], [11], [12]. However, analog based CS cannot be used with most of the existing camera sensors that produces digital outputs (i.e. pixel streams). Digital CS overcomes this problem by performing compressive sensing on the pixel outputs of existing camera sensors.

In this work, we focus on digital CS for image compression. During CS, the signals are compressed using matrix multiplication and the resulting image can be recovered from the compressed image using an optimization process [13]. The CS reconstruction method does not recover the exact signals, but instead recovers signals that are close to or approximated to the original signals. CS based compression uses a measurement matrix for compressing the input signal. The compressed values are a linear combination of the measurement matrix elements. Due to the approximate nature of CS reconstruction, less accurate compressed values can still be recovered. This provides the opportunity to use approximate computing for CS based compression which has previously been shown to be effective in bio-signal processing [14].

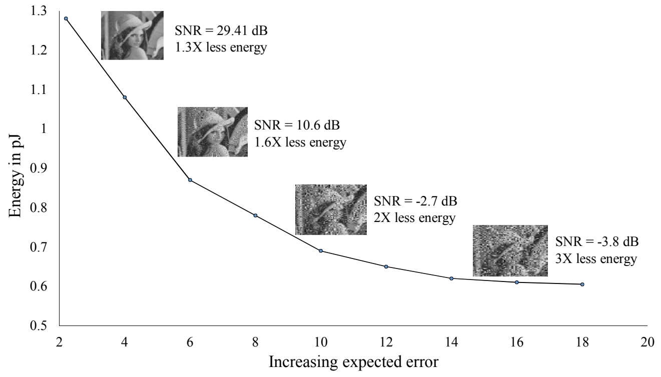


Fig. 1. Output images of approximate FFT architectures with different inexact components. More energy savings can be obtained at expense of lower output image quality [8].

In this paper, we will exploit approximate computing for CS based image compression in order to reduce hardware cost and power consumption. Image compression typically requires many operations due to the large image sizes. For example, CS based compression of 512×512 size image requires 26.7 million multiply and accumulate operations, at a measurement rate of 0.1. As a first step, we developed a library of inexact multipliers using a probabilistic pruning approach. Various internal elements of multipliers are removed to create different instances of inexact multiplier circuits, and the pruning effects on the outputs of the multiplier instances are measured. The pruning of the internal elements also leads to varying power and area savings of the instances. These instances form our inexact multiplier library components. The unpruned multiplier is also included in the library as reference.

The library components are used to replace the exact multipliers in the original CS based image compression architecture. We present a multi-objective optimization approach to judiciously replace the multipliers in the image compression architecture with the inexact library components in order to achieve a good trade off between the compression performance, and power-area savings. Our experiments show that the proposed architecture on 65-nm CMOS technology achieves 43% area and 54% power savings when compared to the original architecture with minimal PSNR degradation.

The rest of the paper is organized as follows. In the following section we provide a brief background on compressive sensing and approximate computing. In Section III, we describe the architecture and operations of the CS based image compression architecture that will be used as our baseline. In Section IV, we introduce our methodology for designing the library of inexact multipliers and propose an optimization technique for choosing suitable library instances for the image compression architecture. Section V presents the experimental results and hardware savings obtained by the proposed architecture. Finally, Section VI concludes the paper.

II. Background

In this section, we briefly review the concepts in compressive sensing and approximate computing.

A. Compressive Sensing

Given a $sparse$-signal, the compressive sensing method acquires sub-Nyquist rate samples from the signal. It compresses N-samples of the sparse signal (X) into M-samples of signal (Y) using a measurement matrix Φ of size $M \times N$, where $M << N$. This method is broadly called CS compression or CS encoding. The equation governing CS compression is shown in Eq. 1, where X is a sparse signal.

$$Y_{M\times 1} = \Phi_{M\times N} X_{N\times 1} \qquad \text{where M} \ll \text{N} \tag{1}$$

An estimate of the original sparse signal X can be recovered from Y using the l_0-optimization. Since l_0-optimization is a NP-complete problem, we can apply Basis-Pursuit (BP) or l_1-optimization as an alternative method for recovering signal. l_1-optimization is defined as:

$$X = arg\ min \parallel X \parallel_1 subjected\ to\ Y = \Phi X \tag{2}$$

We can also apply CS to non-sparse signals, if the non-sparse signals are present in certain domains e.g. Discrete Wavelet, Discrete Cosine etc. The non-sparse signals can be compressed using Eq. 1 and reconstructed using Eq. 3. Ψ in Eq. 3 represents the Discrete Wavelet or Discrete Cosine transform.

$$X = arg\ min \parallel X \parallel_1 subjected\ to\ Y = \Phi\Psi X \tag{3}$$

Compression of 2-D signals like images need to be transformed to 1-D signals before applying CS. The CS reconstruction of images requires more computational power compared to CS compression due to the large value of N. The Block based Compressive Sensing (BCS) presented in [15], splits an image into several small blocks of size $B \times B$ and applies CS compression and reconstruction method on each block. In this paper, we use the BCS method [16] for image compression and a Smoothed Projected Landweber (SPL) method for image reconstruction. The BCS compression of an image block is given as:

$$Y = \Phi_{M\times B^2} X \tag{4}$$

where Y is the compressed block and X is the image block. The sizes of X, Y and Φ-matrix are $B^2 \times 1$, $M \times 1$ and $M \times B^2$ respectively. The measurement rate (MR) is given as:

$$Measurement\ Rate = \frac{M}{B^2} \tag{5}$$

CS compression and reconstruction mainly depends on the measurement matrix Φ. The measurement matrices have to satisfy the Restricted Isometry Property (RIP) [13]. Gaussian random matrices, Partial Fourier matrices and Bernoulli random matrices satisfy the RIP and any of them can be used as measurement matrices. In this paper, we use orthogonal Gaussian random matrix as Φ-matrix for compression of images. The orthogonal random matrices are preferred due to their low computational complexity for CS reconstruction.

B. Approximate Computing

Approximate computing has gained a lot of attention owing to the ever growing demand for power minimization. This approach offers a relaxation on the accuracy of the overall system's output in order to achieve significant gains in power and area. Common strategies for introducing approximate computing in designs include probabilistic pruning, probabilistic logic minimization, and bit-width reduction/truncation. These methods are typically applied to the data-paths of hardware architectures that exhibit higher error tolerance.

Probabilistic Pruning removes elements of a given circuit that do not contribute significantly to the output [17]. On the other hand, *Probabilistic Logic Minimization* selectively changes certain output states of the system so as to reduce the required logic resources [18]. This method is also denoted as *Bit-Flipping*. *Bit-Width Reduction* or *Truncation* is employed to optimize the bit-widths of data-paths with data precision of lesser importance. However, such optimization often leads to additional effort in redesigning the routing architecture [19].

In this work we combine *Probabilistic Pruning* and *Probabilistic Logic Minimization* to design inexact multipliers for our image compression architecture. In addition, we also propose a multi-objective optimization technique to identify suitable inexact levels for each multiplier in the architecture in order to obtain a good trade off between power-area savings and compression performance.

III. Image Compression Architecture

In this section, we present the baseline architecture for digital CS based image compression. The BCS compression in Eq. 4 compresses the image block X of $B \times B$ samples into a compressed block Y of M samples. Before applying the CS compression, the 2D signal X is converted into a 1D signal of size $B^2 \times 1$. The architecture for CS compression, which is based on Eq. 4, is shown in Fig. 2.

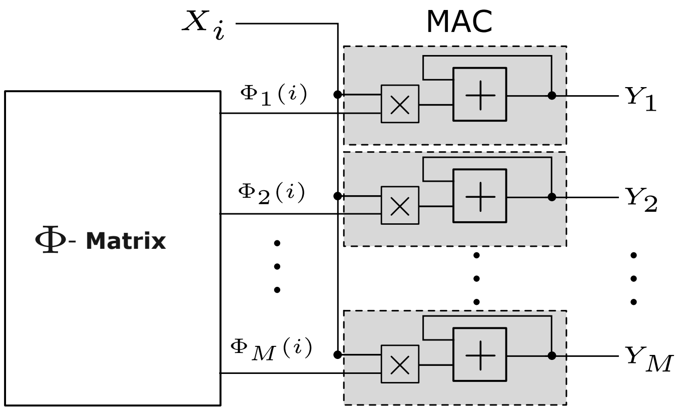


Fig. 2. Architecture of CS compression stage

The architecture in Fig. 2 takes a single image pixel X_i at a time and multiplies it with the corresponding column elements of Φ-matrix. The multiplication results are added to the previous multiply and accumulator (MAC) outputs. The output of the MAC unit at the i^{th} iteration is given as:

$$Y_j^i = Y_j^{i-1} + \Phi_j(i) X_i \tag{6}$$

The term $\Phi_j(i)$ in Eq. 6 represents the element of Φ-matrix in the j^{th} row of the i^{th} column. The MAC units multiply the image pixels with the corresponding matrix elements and add the results to the previous MAC results until the entire image block has been compressed. This process takes B^2 clock cycles for generating the M compressed values of a single image block. All the values of the MAC units are reset to zero before compression of next image block begins. This process is repeated till the compression of an entire image completes.

TABLE I
TOTAL MAC OPERATIONS REQUIRED FOR SINGLE IMAGE BLOCK OF SIZE 32×32 AT DIFFERENT MEASUREMENT RATES (MR)

MR	size of M	Total MAC operations
0.1	102	104448
0.2	205	209920
0.3	307	314368
0.4	410	419840
0.5	512	524288

Each single image block of size $B \times B$ requires MB^2 number of MAC operations for the generation of M compressed samples. Table I shows the number of MAC operations required for a single 32×32 image block at different measurement rates.

In this paper, we consider the block size of 32×32. This provides better PSNR values for reconstructed images. Increasing the block-size will lead to increase in the computational complexity at the reconstruction side. On the other hand, decreasing the block-size reduces the PSNR values for reconstructed images. Hence, we have chosen the optimal block-size to be 32×32. From Table I, we can observe that the minimum number of MAC operations required are in the order of 10^5 for MR 0.1. It can be observed that the MAC operations increases linearly with MR. The image of size 512×512 requires at least 26.7 million MAC operations for generating the compressed samples using CS. The power savings at each MAC operation contributes to the power savings for overall image compression. It is noteworthy that the MSB-bits of compressed samples play a major role in CS reconstruction. This provides the motivation in our work to prune the compression architecture to achieve power-area savings with out affecting the MSB bits of reconstructed signal.

IV. Proposed Method

In this section, we first discuss our methodology to design a library of inexact multiplier units that will aid in pruning the CS compression architecture. Next, we present a Non-dominated Search Genetic Algorithm (NSGA) based multi objective optimization to tune the architecture for achieving a good trade-off between power, area and output quality .

A. Approximate Library Formulation

The library of approximate multipliers is constructed using probabilistic pruning and probabilistic logic minimization.

First according to the *Probabilistic Pruning* method a given circuit is considered as a graph with various gates as nodes and interconnects as edges. The activity for each node in the circuit is estimated based on the path from input to output in which it is present. Based on a its relative influence on the output, a weight is assigned to each node. The nodes are then ranked based on the product of weight and activity. Next, the nodes are removed one at a time starting from the lowest ranked nodes, and the resulting output error of the circuit is estimated. If the estimated error is less than the acceptable error, the pruning is considered to be successful. Otherwise, the node is replaced, and the pruning process is repeated on the next lower ranked node till all nodes have been considered. Each successful pruning create an instance of the inexact multiplier in the library.

Probabilistic Logic Minimization is carried out in a similar manner at the logic level through guided bit flipping of Karnaugh maps to reduce the in-built logic. In this method the logic to be realized by a certain node is further minimized using flipping of certain bits, hence introducing some error. This can lead to either reduced logic or increased logic, resulting in both increase or decrease in power area savings. These can be called as favourable and unfavourable cases. We consider only for the favourable cases. For the input combinations, which occur with less probability, the bit flips would result in less error at the output. Such cases are given more significance and hence stand out a better chance to be part of the library.

The removal of various internal logic as described above, will give rise to different errors at the output and also different power, area savings. We categorized these instances into different inexact levels based on the output degradation. Using the above-mentioned techniques, we created 11 different multiplier instances with increasing magnitude of error at the output. These are termed as inexact levels, $L_1, L_2 \ldots, L_{11}$. L_1 refers to the exact multiplier, L_2 refers to inexact 1, L_{11} refers to inexact 10, so on. The multiplier instances form the members of multiplier library (L_{mult}). All the components (members) in the library are 16 bit multipliers. The normalized power, area savings (with respect to that of the exact multiplier) for each inexact level are shown in Fig. 3.

From the compression architecture in Fig. 2, it can be observed that each MAC unit consists of a multiplier and an adder. In this work, we focus on pruning the multipliers only, since the error introduced by pruning the adders exceeds the acceptable level. Each of the multipliers in the 'M' MAC units can be replaced with a particular component from L_{mult}, as shown in Fig. 4. In the next subsection, we describe an approach to judiciously identify suitable multiplier instances for the compression architecture.

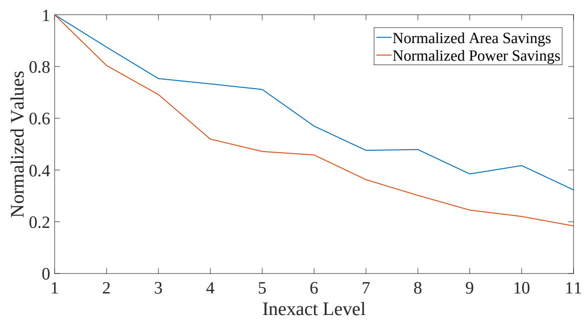


Fig. 3. Normalized area and power savings of the compression architecture obtained using various inexact multipliers. Each instance of the architecture shown uses multipliers with the same inexact level.

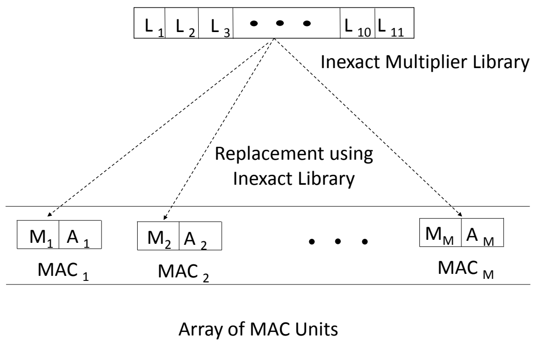

L_1 L_2 L_3 • • • L_{10} L_{11}
Inexact Multiplier Library
Replacement using Inexact Library
M_1 A_1
MAC $_1$
M_2 A_2
MAC $_2$
• • •
M_M A_M
MAC $_M$
Array of MAC Units

Fig. 4. A library of multipliers created using *Probabilistic Pruning and Probabilistic Logic Minimization[19]*, are used to replace the multipliers in the MAC units of the compression architecture.

B. Allocating Inexact Multipliers to Compression Architecture

The library (L_{mult}) described in the previous subsection contains k number of multiplier instances ($k = 11$), each obtained through *probabilistic pruning* and *probabilistic logic minimization*. Each of the k instance have a corresponding power-area savings, along with the error value. Note that the behaviour of these k components are characterized by using a large database of random samples as inputs and observing their respective outputs. The error associated with each output of the inexact instance is then calculated based on the difference with the exact component. The power, area and error characteristics of each instance are used for allocating the multiplier instances to the compression architecture.

For a particular measurement rate (say 0.3) of the compression architecture, the number of multipliers are fixed. Each multiplier can be implemented using one of the levels (instances) of inexactness defined in the library L_{mult}. The set of inexact levels for the multipliers can be regarded as a level vector ($\overrightarrow{Level}$) which corresponds to a particular area, power savings and error at the output. In order to make judicious choice in the selection of inexact levels for each multiplier, we use a multi-objective optimization. The two objectives we focus in this work are power-area product (PAP), maximum error ($Maxerror$) at the ouput of the CS architecture. We aim to simultaneously minimize simulataneously PAP and $Maxerror$. A non dominated sorting based genetic algorithm (*NSGA-II*) [20] is used to carry out this optimization. Compared to other genetic algorithms, (*NSGA-II*) has faster convergence and elitism is maintained during selection process. Hence we chose this optimization method. We calculate PAP for a given level vector $\overrightarrow{Level}$ using a function f_{pap} and $Maxerror$ by another function f_{err}. Both these functions are written as different sub routines.

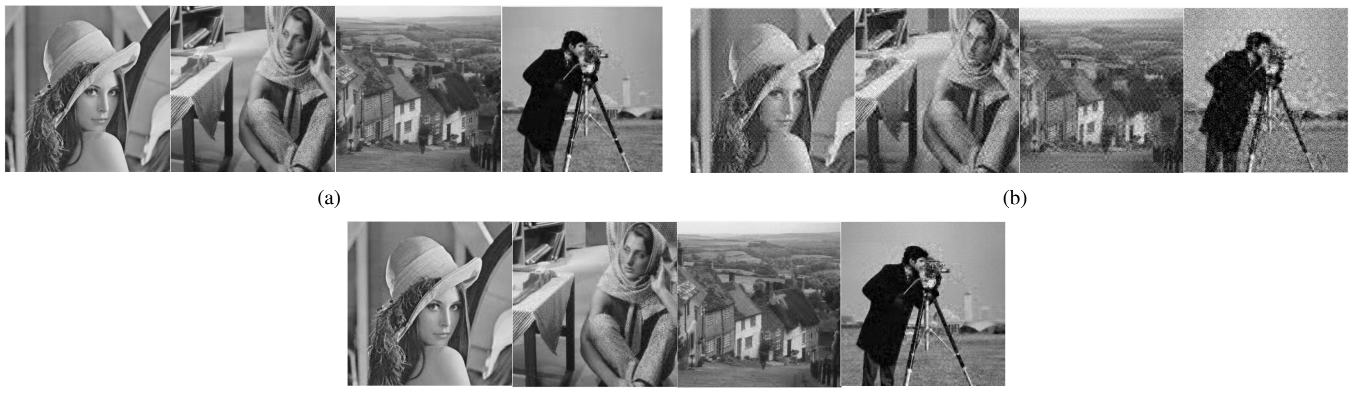
Hence, the problem of finding an optimum level vector $\overrightarrow{Level_{opt}}$ for entire array of multipliers *Opt_Conf* can be defined as :

$$
\begin{aligned}
Opt\_Conf\{\overrightarrow{Level_{opt}}\} = & minimize[PAP = f_{pap}(\overrightarrow{Level})], \\
& minimize[Maxerror = f_{err}(\overrightarrow{Level})], \\
& subject\ to\ \overrightarrow{Level_{exact}} \subseteq \overrightarrow{Level_{opt}} \subseteq \overrightarrow{Level_{max}}, \\
& and\ PAP_{opt} \leq PAP_{TH}, err_{opt} \leq err_{TH} \\
& for\ some\ constants\ PAP_{TH}, err_{TH}
\end{aligned}
$$

The *NSGA-II* optimization gives an optimum level vector ($\overrightarrow{Level_{opt}}$), which yields a power-area product (PAP_{opt}) and error (err_{opt}) satisfying the threshold requirements (PAP_{TH}, err_{TH}). $\overrightarrow{Level_{max}}$ refers to all multipliers realized using L_{11} and $\overrightarrow{Level_{exact}}$ refers to all multipliers using L_1. $\overrightarrow{Level_{opt}}$ is further used for realizing the optimum MAC architecture.

V. Experimental Results

We performed the CS compression using original and various inexact configurations with varying measurement rate from 0.2 to 0.9. The CS architecture in Fig. 2 takes an image pixel of size 8-bits and multiplies it to the corresponding Φ-matrix element of size 8-bits. This is added to the previous output value. The elements of Φ-matrix range from 0 to 0.99.



(a) (b) (c)

Fig. 5. Reconstructed Images (Lenna, Barbara, Gold hill, Cameraman) with measurement rate of 0.3 at various inexact levels (a) level 1 (b) level 11 (c) level 6 (optimum case)

Hence, a maximum of 16-bits are sufficient for representing the compressed samples Y. Fig. 5 shows the reconstructed images with original (level 1), worst case approximate (level 11) and optimum approximate (level 6) compressed values with block-size of 32×32 and measurement rate of 0.3.

It can be seen from Fig. 5, that there is a significant loss of quality in the reconstructed images with higher inexactness (level 11) compared to the quality of images in optimum case (level 6). This establishes the need for judicious pruning for realizing approximate architectures. Fig. 6 compares the peak signal-to-noise ratio (PSNR) in dB for different images over different inexact levels. Maximum degradation of PSNR, occurs for 'Lenna' at inexact level 11. From the figure, it can be observed that PSNR values do not decrease linearly with higher inexact levels. This is due to the fact that the inexact levels are assigned based on the standard deviation error. This acts as a generalized yardstick for the library. However the mean error of the levels may have a different trend, which result in variations of PSNR trend. By and large we can observe a trend of decreasing PSNR in higher inexact levels.

The approximate library instances are described using hardware description languages (i.e. VHDL) and synthesized using Synopsys Design Compiler. The error estimate model is designed in MATLAB. The entire design process is carried out using technology process TSMC 65nm. The supply voltage for technology node is obtained from the corresponding foundry specifications. The performance advantage obtained by approximate components are validated in two circumstances namely maximum operating frequency and minimum applicable power. The normalized gains for various error metrics of 16-bit array multiplier are recorded. Since the approximate components are used in the data path, we fixed the bit width to be 16.

We have used the MATLAB optimization tool box for carrying out the multi objective optimization. The parameters for optimization are chosen as described in [20]. Standard two point crossover and bit string format for chromosome are chosen. Maximum error caused by the inexact architecture and the power area product(pap) are considered as two objectives. For a particular case (measurement rate = 0.3, sample 'Cameraman') the pareto curve is shown in Fig. 7. By definition [20], all the points existing on the pareto curve are as good as others. However, for the optimum case, we chose a point at the knee of the curve and consider it as our 'Characteristic point'. This point gives a very good *PAP* savings and beyond this the max_error increases steeply. The optimum power and area savings are estimated for this point only.

Changing inexact levels at a granularity of individual multiplier as well as at entire array of multipliers is also explored. Rigorous analysis leads to level 6 as the optimum case. This case produced a PSNR degradation near to 1 dB (average over various images). We estimated the overall power and area for the inexact architecture, to observe affect of approximation at system level. We considered multiplier and adder blocks to be the most important power consuming blocks. In this work, as mentioned earlier we prune only multipliers because the error introduced by inexact adders is too huge compared to their returns in terms of power, area. Hence, for overall estimation the power, area for adder blocks is considered same as in the exact case.

The optimal architecture showed an overall 43% area improvement and 54% power improvement which demonstrate the usefulness of judicious realization inexact circuits. The power area product is improved by $3.84\times$ when compared to the exact architecture.

VI. Conclusions

This work presents a novel approach for minimizing the area utilization and power consumption of compressive sensing based image compression architecture with marginal PSNR degradation. We designed an inexact MAC architecture for realizing the compression circuit. Our approach is validated over different standard inputs, and achieves 54% and 43% improvement in overall power and area respectively, with only a slight degradation in PSNR. In future, we plan to extend this work for inexact realization of the reconstruction architecture.

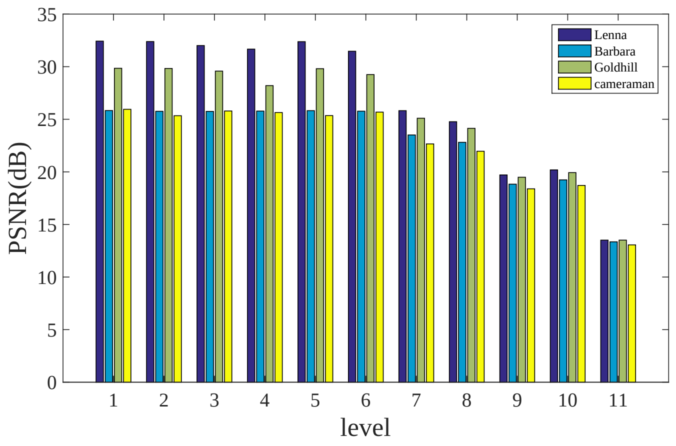


Fig. 6. Variation of PSNR (dB) values of reconstructed images using multipliers belonging to original and various approximate levels

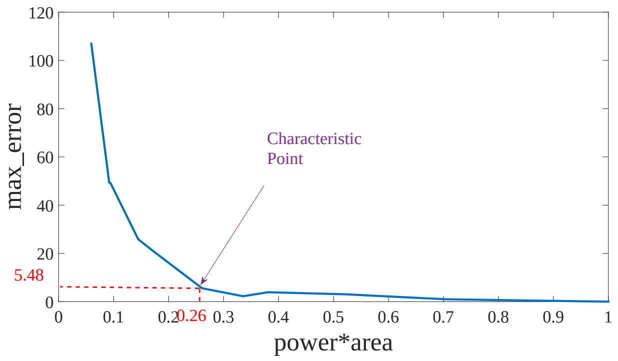


Fig. 7. Pareto curve showing Power Area Product (PAP) and maximum error incurred for 'Cameraman' sample at subrate 0.3. The characteristic point from the curve is chosen to realize optimum architecture. It has maximum error of 5.48 and PAP 0.26× (power = .46× , area = .57×)

References

[1] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *2013 18th IEEE European Test Symposium (ETS)*. IEEE, 2013, pp. 1–6.

[2] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," in *ACM SIGPLAN Notices*, vol. 47, no. 4. ACM, 2012, pp. 301–312.

[3] I. Chong, H.-Y. Cheong, and A. Ortega, "New quality metric for multimedia compression using faulty hardware," in *Intl Workshop on Video Processing and Quality Metrics for Consumer Electronics*. Citeseer, 2006.

[4] S. H. Kim, S. Mukohopadhyay, and W. Wolf, "Experimental analysis of sequence dependence on energy saving for error tolerant image processing," in *Proceedings of the 2009 ACM/IEEE international symposium on Low power electronics and design*. ACM, 2009, pp. 347–350.

[5] C. Alvarez, J. Corbal, and M. Valero, "Dynamic tolerance region computing for multimedia," *IEEE Transactions on Computers*, vol. 61, no. 5, pp. 650–665, 2012.

[6] M. Imani, D. Peroni, A. Rahimi, and T. Rosing, "Resistive cam acceleration for tunable approximate computing," *IEEE Transactions on Emerging Topics in Computing*, 2016.

[7] M. Imani, Y. Kim, A. Rahimi, and T. Rosing, "Acam: Approximate computing based on adaptive associative memory with online learning." in *ISLPED*, 2016, pp. 162–167.

[8] A. Lingamneni, K. K. Muntimadugu, C. Enz, R. M. Karp, K. V. Palem, and C. Piguet, "Algorithmic methodologies for ultra-efficient inexact architectures for sustaining technology scaling," in *Proceedings of the 9th conference on Computing Frontiers*. ACM, 2012, pp. 3–12.

[9] E. J. Candès and M. B. Wakin, "An introduction to compressive sampling," *IEEE signal processing magazine*, vol. 25, no. 2, pp. 21–30, 2008.

[10] F. Chen, A. P. Chandrakasan, and V. M. Stojanovic, "Design and analysis of a hardware-efficient compressed sensing architecture for data compression in wireless sensors," *IEEE Journal of Solid-State Circuits*, vol. 47, no. 3, pp. 744–756, 2012.

[11] M. Dadkhah, M. J. Deen, and S. Shirani, "Block-based CS in a CMOS image sensor," *IEEE Sensors Journal*, vol. 14, no. 8, pp. 2897–2909, 2014.

[12] Y. Oike and A. El Gamal, "CMOS image sensor with per-column $\sigma\delta$ adc and programmable compressed sensing," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 1, pp. 318–328, 2013.

[13] E. J. Candes, "The restricted isometry property and its implications for compressed sensing," *Comptes Rendus Mathematique*, vol. 346, no. 9, pp. 589–592, 2008.

[14] D. Bortolotti, H. Mamaghanian, A. Bartolini, M. Ashouei, J. Stuijt, D. Atienza, P. Vandergheynst, and L. Benini, "Approximate compressed sensing: ultra-low power biosignal processing via aggressive voltage scaling on a hybrid memory multi-core processor," in *Proceedings of the 2014 international symposium on Low power electronics and design*. ACM, 2014, pp. 45–50.

[15] L. Gan, "Block compressed sensing of natural images," in *2007 15th International conference on digital signal processing*. IEEE, 2007, pp. 403–406.

[16] S. Mun and J. E. Fowler, "Block compressed sensing of images using directional transforms," in *2009 16th IEEE international conference on image processing (ICIP)*. IEEE, 2009, pp. 3021–3024.

[17] A. Lingamneni, A. Basu, C. Enz, K. V. Palem, and C. Piguet, "Improving energy gains of inexact dsp hardware through reciprocative error compensation," in *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*. IEEE, 2013, pp. 1–8.

[18] A. Lingamneni, C. Enz, K. Palem, and C. Piguet, "Synthesizing parsimonious inexact circuits through probabilistic design techniques," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 12, no. 2s, p. 93, 2013.

[19] A. Lingamneni, C. Enz, J.-L. Nagel, K. Palem, and C. Piguet, "Energy parsimonious circuit design through probabilistic pruning," in *2011 Design, Automation & Test in Europe*. IEEE, 2011, pp. 1–6.

[20] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.