Optimal Design of JPEG Hardware Under the Approximate Computing Paradigm

Farhana Sharmin Snigdha[1], Deepashree Sengupta[1], Jiang Hu[2] and Sachin S. Sapatnekar[1],
[1]Department of ECE, University of Minnesota*
[2]Department of ECE, Texas A&M University
{sharm304,sengu026,sachin}@umn.edu, jianghu@ece.tamu.edu

ABSTRACT

JPEG compression based on the discrete cosine transform (DCT) is a key building block in low-power multimedia applications. We use approximate computing to exploit the error tolerance of JPEG and formulate a novel optimization problem that maximizes power savings under an error budget. We analyze the error propagation sensitivity in the DCT network and use this information to model the impact of introduced errors on the output quality. Simulations show up to 15% reduction in area and delay which corresponds to 40% power savings at iso-delay.

Keywords

Approximate computing; Low-power design; JPEG; Image compression; Nonlinear optimization

1. INTRODUCTION

Energy-efficiency demands have driven interest in designs that leverage the inherent error resilience of many multimedia applications [1–4]. In this context, approximate computing uses simplified, lower power hardware operations with controlled errors, using logic or architecture modification [1, 2], through voltage overscaling [4], time starvation [5], etc.

A typical target for this optimization is in the domain of digital signal processing (DSP), which is a key functionality in multimedia processors. Typical DSP operations used in image and video processing units include image compression, filtering, and reconstruction. This paper develops an optimized approximate computing solution for image compression using widely-used JPEG compression scheme.

A core building block for JPEG compression is the discrete cosine transform (DCT). Several algorithms for implementing the DCT have been proposed, and they largely aim to reduce the number of computations *without approximations* by altering the architecture or/and logic simplifications [6–8]. These exploit the periodic symmetries of cosine terms in the DCT flow graph. However, even for a moderate-size image, the number of computations is large [9].

A few prior methods have explored approximations in JPEG compression, but largely perform optimizations in an *ad hoc* style. Some approaches use truncation, such as dynamic bit width [10] and dynamic range reduction [3], while others uniformly approximate all adders in the DCT [1]. Other related approaches [11, 12] target general RTL structures and do not exploit the structure of JPEG.

We address the problem of building optimized JPEG compression hardware by optimally approximate arithmetic units within the hardware to reduce the power consumption under a user-specified error budget. JPEG compression is so widely used that it is worthwhile to build a very specific optimization technique for this hardware. At the same time, we develop optimization principles that we believe will be applicable for optimizing widely-used general DSP blocks, such as those that implement FFTs or correction algorithms. Our formulation finds the sensitivity of error at the output node to an error introduced at an internal node, and proposes a novel technique for error variance propagation within the graph. Finally, we propagate the error budget at the output to internal nodes and use the notion of sensitivity to build a design that meets this budget.

2. JPEG DECOMPOSITION

The JPEG compression scheme is based on a 2-dimensional (2D) frequency-domain DCT, followed by a quantization step, as illustrated in Figure 1. The method operates by dividing an image, I_{image}, into 8×8 pixel blocks and compressing each block separately. Each such block constitutes the 8×8 initial matrix, I, which will be referred to as one *frame*. For example, a typical image of size 512×512 pixel consists of 4096 frames. Each element of the matrix, I, has a value in the range of 0 to 255, but the DCT operation is performed on a shifted matrix, M, obtained from I by subtracting 128 from each entry in I. The result of compressing M is an 8×8 matrix, C. The JPEG compression sequence has two major steps, described in the rest of this section.

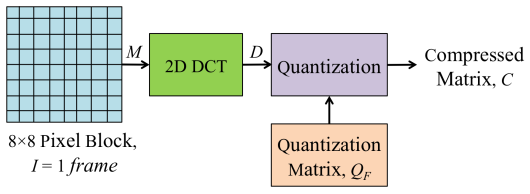

Figure 1: JPEG encoder block diagram.

*This work was supported in part by the NSF under award CCF-1525925.

DAC '16, June 05-09, 2016, Austin, TX, USA

http://dx.doi.org/10.1145/2897937.2898057

2.1 The DCT Stage

2.1.1 Functionality

Each frame, represented by M, undergoes an 8-point 2D DCT. The DCT coefficient matrix, D, is given by

$$D(i,j) = \frac{1}{4}U(i)U(j)\sum_{x=0}^{7}\sum_{y=0}^{7} M(x,y) \times \cos\left(\frac{(2x+1)i\pi}{16}\right) \times \cos\left(\frac{(2y+1)j\pi}{16}\right), \; 0 \le i,j \le 7 \quad (1)$$

where i [j] represent the horizontal [vertical] spatial frequencies. The scaling factor, $U(k)$, is given by

$$U(k) = \begin{cases} \frac{1}{\sqrt{2}} & , \text{if } k = 0 \\ 1 & , \text{if } k > 0 \end{cases} \quad (2)$$

2.1.2 Hardware Implementation

The separable property of the DCT allows the 2D operation to be decomposed in two sequential 1D DCT operations along the row and column of M, as shown in Figure 2. We denote the first and second sets of 1D DCT blocks as Layer 1 and Layer 2, respectively. Each layer contains eight individual 1D DCT blocks, so that over the two layers, 16 such 1D DCT blocks are required. For each block, i, in the first layer, its input comes from the i^{th} column of M, denoted by M_i, $i \in \{0, \cdots, 7\}$. The outputs of the first layer are fed as inputs to the second layer after a transpose operation.

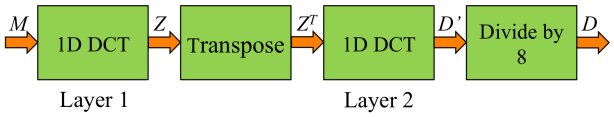

Figure 2: Decomposition of 2D DCT architecture into 1D DCT structures.

The periodic structure of cosine terms in (1) can be exploited to efficiently implement fast DCT operations by combining several additions and multiplications, and a number of fast 1D DCT algorithms have been proposed [6–8]. Of these, the Loeffler 1D DCT algorithm [8], which requires 11 multiplications and 29 additions, achieves the theoretical lower bound for multiplications in 8-point DCT, and we have used it as a basis for our work. The output Z_i of each 1D DCT block using Loeffler's algorithm is

$$Z_i = \sqrt{2}\sum_{x=0}^{7} U(i)\, M_i \cos\frac{(2x+1)\, i\pi}{16} \quad (3)$$

where M_i is as defined above. An extra $\sqrt{2}$ term is incorporated to obtain Z_0 and Z_4 without any multiplication. When this implementation is used for the 1D DCT in each layer in Figure 2, a final divide-by-8 stage is required, as shown in the figure, in order to translate the output D' of Layer 2 to an output D consistent with (1). Algorithm 1 shows the process of computing JPEG compressed matrix for a single frame using the Loeffler fast DCT technique, and compression corresponds to steps 1 through 8.

The Loeffler 1D DCT has four primary stages and can be represented by a butterfly diagram shown in Figure 3(a), where the internal nodes of the butterfly are as described in Figure 3(b). It is important to note that *all multipliers perform constant multiplications*, where the constant input is either $\sqrt{2}$, a_n, $(b_n - a_n)$, or $(a_n + b_n)$. These values are fractions, but practically, these are stored as fixed-bit integers

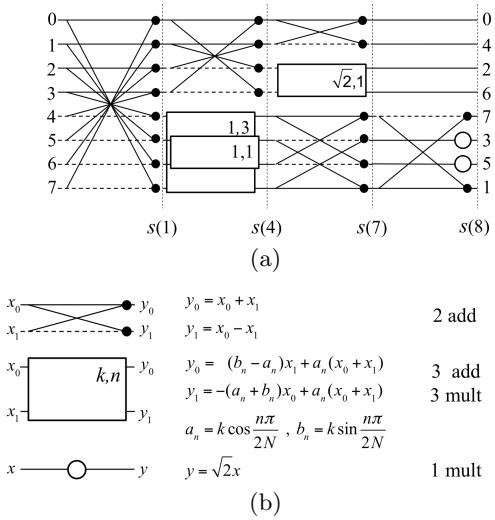

Figure 3: (a) Butterfly network for an 8-point 1D DCT. (b) Computations at internal nodes.

by multiplying them by 2^7, and this scaling is accounted for in the final output through appropriate shift operations.

Based on the equations shown in Figure 3(b), it can be shown that the 1D butterfly shown in Figure 3(a) requires total 29 additions and 11 multiplications [8]. Thus, for each frame, the net computation over Layers 1 and 2 involves $16 \times 11 = 176$ multiplications and $16 \times 29 = 464$ additions. All calculations are performed using 32-bit signed arithmetic. The divide-by-8 step is achieved using shift operations.

Algorithm 1 Finding the JPEG Compression Matrix, C

INPUT: 8×8 Image pixel matrix, M
OUTPUT: 8×8 JPEG compressed matrix, C.
METHOD:

1: **for** $i \leftarrow 1$ to 8 **do** ▷ Row operation
2: $\quad Z(i,:) = f_{1D,DCT}(M_i)$
3: **end for**
4: **transpose**
5: **for** $i \leftarrow 1$ to 8 **do** ▷ Column operation
6: $\quad D(i,:) = f_{1D,DCT}(Z_i)$
7: **end for**
8: $D(i,j) >> 3$ ▷ Divide by 8
9: **return** $C(i,j) = \text{round}\left[\frac{D(i,j)}{Q_F(i,j)}\right]$ ▷ Quantization step

2.2 The Quantization Stage

As illustrated in Algorithm 1, the DCT coefficient matrix, D, undergoes a lossy compression step in the final step in line 9 as it performs the operation,

$$C(i,j) = \text{round}\left[\frac{D(i,j)}{Q_F(i,j)}\right] \quad (4)$$

i.e., each element in the DCT coefficient matrix, $D(i,j)$, is individually divided by the corresponding element of a quantization matrix, $Q_F(i,j)$, and rounded to the nearest integer to form the quantized result, $C(i,j)$, of JPEG compression.

The subscript F of the quantization matrix, Q_F, is a quality factor ranging from 1 to 100, where 1 corresponds to the lowest quality. Commonly-used values of F are 50 and 90. The rounding of the results is a key step where some information of the image is discarded. At a lower quality factor,

F, the divisors are higher, implying that higher compression ratio is achieved [13]. Further, the key DCT coefficients are close to $D(0,0)$, and the elements of Q_F progressively increase as we move further away from the $(0,0)$ element.

3. APPROXIMATE COMPUTING IN JPEG

The amount of computation and power dissipation in a DCT is significant: for a moderate-sized 512×512 (256K) image with 4096 frames, the fast DCT requires nearly 10^6 multiplications and 2×10^6 additions. The large number of operations, along with the error resilience of JPEG to approximation, motivates us to explore approximate computing.

We represent the structure of the Loeffler 1D DCT in Figure 3(a) as a directed acyclic graph (DAG), where each node in the graph represents arithmetic operations such as add/subtract or multiply. Along with the four primary stages in Loeffler 1D DCT in Figure 3(a), we show the computations of Figure 3(b) to illustrate a total of eight stages, $s(i), i = 1, \cdots, 8$, between the input and output in Figure 4.

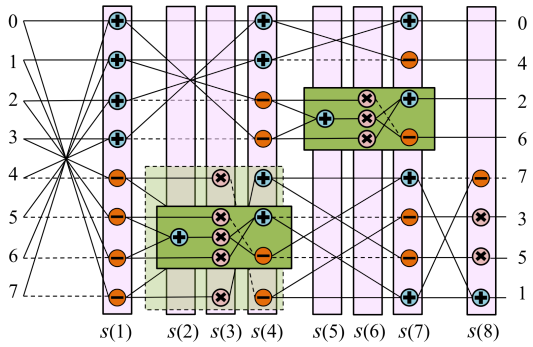

Figure 4: A DAG model of the Loeffler DCT.

Adders and multipliers are simply arrays of full adders (FA). Multiplication is implemented using a 4 : 2 compressed carry save adder (CSA) based multiplier with m least significant bits (LSB) approximated for each node in the DAG. Addition is implemented using 32-bit ripple carry adders. We have implemented the five transistor level full adder designs proposed in [1] in a 32-bit multiplier and concluded that the "Approximate 5" adder (henceforth abbreviated as *appx*5) provides the best area and one of the lowest errors, and hence we use this structure in our implementation.

4. PROPOSED APPROXIMATE SCHEME

The total error in the JPEG compression scheme is a function of the approximation scheme used at each node. In this section, we develop an algorithm for finding the most effective approximation by developing error models and incorporating them into an optimization framework. The optimization is carried out over the two layers in Figure 2, each containing eight 1D Loeffler DCT structures shown in Figure 5, each of which processes one column of the M matrix.

However, optimizing the number of approximate bits at the granularity level of an individual 1D DCT block implies that each block could have a different structure, and this surrenders the major advantage of regularity that is exploited in layout optimization of DSP blocks. Therefore, we choose to restrict the optimization so that within each layer, all eight 1D DCT blocks are identical, allowing for easier design and layout. Optimization variables in layer $l \in \{1, 2\}$ are the number of approximate bits in

- adder i, $i = 1, \cdots, 29$, denoted as a_i^l.
- multiplier j, $j = 1, \cdots, 11$, denoted as m_j^l.

Thus, the total number of variables is $2 \times (29 + 11) = 80$. In fact, we find that doubling the optimization variables, i.e. two sets of variables for each layer only result in very small changes in pre-layout estimates of power as shown in Section 5, vindicating our choice.

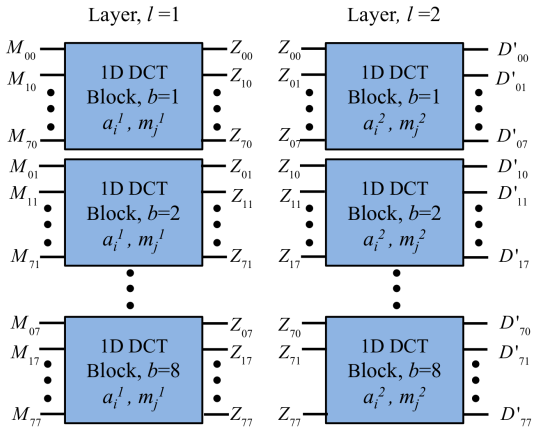

Figure 5: The 2D DCT using 16 1D DCT blocks.

The errors introduced due to approximating some bits in these adders and multipliers accumulate at the output of the JPEG compression engine and affect the quality of the resultant image. The error at each node within the DAG has a different contribution to the total error at the final output. We propose a scheme based on the sensitivity of each node to the total output error to optimize the hardware, subject to a user-specified error budget that is typically application-dependent. Our methodology can be broadly divided into:

- **Computing the error statistics of each unit module:** Characterization of the error mean and variance for the unit arithmetic units (adders and multipliers).
- **Sensitivity analysis of the DCT block:** Calculation of the sensitivity of each node in a 1D DCT block to the error at the output stage.
- **Nonlinear optimization:** Formulation of the problem as a nonlinear optimization to obtain a_i^l and m_j^l.

4.1 Computing Error Statistics

Errors in Adders: For the approximate FA design *appx*5, the error x from approximating α LSBs can take on 2^α different values from the set $\{0, \pm 1, \pm 2, \cdots, \pm\left(2^{\alpha-1} - 1\right), -2^{\alpha-1}\}$. The approximation scheme in the adder is concentrated in the lower LSBs. Over the large number of frames processed by an adder, it can be reasonably assumed that the lower LSBs of the input have equal probability of ones and zeros. Using this uniform distribution of error values, i.e., $p_x = 1/2^\alpha$, we find the error mean and variance:

$$\mu_{add}(\alpha) = -0.5 \quad (5)$$

$$\sigma^2_{add}(\alpha) = \frac{4^\alpha - 1}{12} \quad (6)$$

Note that the error mean for adder is independent of number of approximated bits, and for all practical purposes it is zero: even for a 1 LSB approximation, the variance is 1 unit, so that 3σ is much greater than the mean; this effect is even stronger when more LSBs are approximated.

Errors in Multipliers: Each multiplier has one constant node and one variable node, as described in Section 2.1.2.

Assuming a uniform input distribution, the error statistics of multipliers are analytically modelled using the MATLAB curve fitting toolbox as a function of the α approximated LSBs. The similarity of the coefficients of all the equations led us to choose one single equation shown below:

$$\sigma^2_{mul}(\alpha) = 0.6135\, e^{1.38\alpha} \tag{7}$$

As before, the error mean is virtually zero.

4.2 Sensitivity Analysis of the DCT Block

The computation at each node of a 1D DCT block is a linear combination of the results of its immediate predecessor stage node. If we denote the result at node n of stage s, as $\eta_{n,s}$,

$$\eta_{n,s} = \sum_{k=1}^{8} W_{k,n,(s-1)} \times \eta_{k,(s-1)}, 1 \le n, s \le 8 \tag{8}$$

where $W_{k,n,(s-1)}$ is the weight of node k of the previous stage, $(s-1)$ for node, n, of stage s. The value of $W_{k,n,(s-1)}$ can be obtained from the structure of the network, shown in Figure 4. We use $\zeta_{n,s,k}$ to denote the sensitivity of node k of stage $(s-1)$, at node n of stage s. From (8),

$$\zeta_{n,s,k} = \frac{\partial \eta_{n,s}}{\partial \eta_{k,(s-1)}} = W_{k,n,(s-1)} \qquad \forall n,s,k \in \{1,\cdots,8\} \tag{9}$$

Given the variances of nodes at predecessor stages, the error variance at a node can be obtained by weighting these variances by the sensitivity, plus the error generated at the node due to the use of an approximate operator. The error variance, $\sigma^2_{n,s,b,l}$, for node n, stage s, block b, layer l is

$$\sigma^2_{n,s,b,l} = \sum_{k=1}^{8} \zeta^2_{n,s,k} \times \sigma^2_{k,(s-1),t,l} + \sigma^2_{op,nsl} \tag{10}$$

where

$\sigma^2_{op,nsl}$ = Generated error variance for operation at node, n, at stage, s, for layer, l

$\sigma^2_{n,0,b,l}$ = Input error variance for block, b, of layer, l

The values of $\sigma^2_{op,nsl}$ can be obtained from (6) and (7).

Based on the application requirement, a maximum error, $\delta_e = \mu_{budget} + 3\sigma_{budget}$, is specified by the user at the output of the compression stage, corresponding to the allowable error in each pixel of the compressed image, C, where μ_{budget} and σ_{budget} are the mean and standard deviation of the maximum error budget, respectively. Using similar arguments as before, the mean of δ_e is close to zero. Therefore, the relation between output error variance budget, σ^2_{budget}, and the maximum error budget, δ_e, is shown below

$$\sigma^2_{budget} = \left(\frac{\delta_e}{3}\right)^2 \tag{11}$$

4.3 The Nonlinear Optimization Formulation

The number of FAs associated with k approximate output LSBs in an array multiplier, for reasonable k, is quadratic in k. For an adder, each approximate output bit translates to one approximate FA. Let the number of approximate FAs in an adder and multiplier be denoted by $f_a(k)$ and $f_m(k)$, respectively. Then

$$f_a(k) = k\ ;\ f_m(k) = \frac{k(k-1)}{2} \tag{12}$$

Thereferfore, the total number of FAs that are approximated can be represented as a function of the number of approximated bits associated with each adder or multiplier. Over both layers, this is represented as:

$$\lambda_{appx} = 8 \times \left(\sum_{l=1}^{2} \sum_{i=1}^{29} f_a(\hat{a}^l_i) + \sum_{l=1}^{2} \sum_{j=1}^{11} f_m(\hat{m}^l_j) \right) \tag{13}$$

where double summations are used to incorporate total numbers of adders and multipliers in single 1D DCT block for each layer, l. The multiplying factor of 8 captures the notion that the same approximation is used within each of the 8 1D DCT blocks in layer, l, to preserve layout regularity. We define a metric, "Percentage FA savings" (PFA), to quantify the percentage of the total FAs, which is defined as:

$$PFA = \left(\frac{\lambda_{appx}}{\lambda_{total}}\right) \times 100 \tag{14}$$

To obtain λ_{total}, we see that the DCT hardware requires 464 adders and 176 multipliers, so that for a 32-bit computation,

$$\lambda_{total} = 464 \times f_a(32) + 176 \times f_m(32) = 102,144 \tag{15}$$

Based on the relations in Sections 4.1 and 4.2, we formulate an optimization problem that maximizes the savings due to approximation, subject to a specified bound on the error introduced in the compressed image.

The precise formulation of the optimization problem is:

$$\max \quad \sum_{l=1}^{2} \sum_{i=1}^{29} a^l_i + \sum_{l=1}^{2} \sum_{j=1}^{11} \frac{m^l_j\left(m^l_j - 1\right)}{2} \tag{16}$$

s. t. (a) $\forall n,s,b \in \{1,\cdots,8\}, l \in \{1,2\}$,

$$\sigma^2_{n,s,b,l} = \sum_{k=1}^{8} \zeta^2_{n,s,k} \times \sigma^2_{k,(s-1),t,1} + \sigma^2_{op,nsl}$$

(b) $\sigma^2_{n,0,b,1} = 0 \quad \forall n,b \in \{1,\cdots,8\}$

(c) $\sigma^2_{n,0,b,2} = \sigma^2_{n,8,b,1} \quad \forall n,b \in \{1,\cdots,8\}$

(d) $\sigma^2_{n,8,b,2} \le \left(\frac{8\delta_e}{3}\right)^2 \quad \forall n,b \in \{1,\cdots,8\}$

The objective function corresponds to maximizing λ_{appx}, which represents the total hardware savings in the system. The constraints correspond to the error variances, and are represented stage by stage. Constraint (a) represents the relationship (10) between the error variance from one stage to the next in each layer of the DCT network. Constraints (b) and (c), respectively, state that there is no error at the input of layer 1 of the DCT on the compression side when M is presented, and that the error of stage 8 of layer 1 is passed on to layer 2. The variance constraint (d) at the output corresponds to (11), and the new factor of 8 in this inequality accounts for the divide-by-8 stage after Layer 2.

Since the error variance equations of adder and multiplier are nonlinear, as shown in (6) and (7), both the objective and constraints are nonlinear. The decision variables, a^l_i and m^l_j, for this optimization are integer-valued. Although a mixed integer linear programming problem is generally computationally expensive, there are two factors in our favor. First, the number of variables is small enough that a solution is realistic. Second, since this is a one-time solution at design time, computation time is not critical. In practice, we find that the problem is solved in about 1 hour for our test cases.

5. RESULTS

We synthesized the RTL of the JPEG hardware using the 45nm Nangate library in Synopsys Design Vision and all simulations have been performed at the typical process corner with $V_{dd} = 1.1$V and temperature, T = 25^{o}C. We have

used benchmark images [14] such as Lena, Pepper_gray, Boat, Arctichare, and Gray21.512, to exercise our algorithm.

We used the mixed integer nonlinear problem solver, KNITRO [15] to solve (16). The CPU times for optimization were of the order of 1 hour in a 2.6 GHz Intel Core i5 CPU with 8Gb RAM and 64-bit OS X. The optimal $\hat{a}_i^l$ and $\hat{m}_j^l$ values obtained from KNITRO for the JPEG hardware were hard-coded into the RTL and synthesized to obtain the area, power and delay of the resulting approximate hardware.

We evaluate the PSNR after compression and reconstruction of the image to reflect the true impact on JPEG accuracy. The reconstruction of the compressed image follows a procedure analogous to compression, except that the inverse DCT (IDCT) technique is used instead of the DCT to reconstruct the image matrix, R. The IDCT can also be implemented using a butterfly, and incurs similar computation as the compression step, but we use only exact computations during this step to evaluate the quality of our algorithm.

The difference between the initial image, I, and the reconstructed image, R, determines the compression quality, measured in terms its peak signal to noise ratio (PSNR). The PSNR of the JPEG compressed image largely depends on the quantization matrix, Q_F. If I_x and R_x are the x^{th} frame of the original and reconstructed image matrices, respectively, then in an image with N frames, the $PSNR$ is:

$$PSNR = 10 \log \left(\frac{\max \left(I_{image}^2 \right)}{MSE} \right) \tag{17}$$

where $MSE = \frac{1}{N} \sum_{x=1}^{N} (I_x - R_x)^2$. We obtained the PSNR of the JPEG compressed images using a MATLAB simulation. The base case for PSNR uses quality factor, $F = 90$ with no approximation. Relative to the base case, we compared the PSNR difference, $\Delta PSNR$, for several images, for various F, with or without approximations.

We plot the PFA as mentioned in (14), power and area savings obtained from a synthesized hardware implementation along with $\Delta PSNR$ vs. the error variance budget, σ_{budget}^2, for two values of quality factors, $F = 90$ and $F = 50$ in Fig. 6(a) and 6(b), respectively. The $\Delta PSNR$ values are plotted on the left axis using bars while the percentage savings for various quantities (area, power and PFA) are plotted on the right axis using points in each graph. The graphs of percentage area savings and PFA correlate well due to the strong correspondence between FA savings and area savings. A good estimation for percentage change in switching capacitance, C_{sw}, can also be obtained from PFA.

At $\sigma_{budget}^2 = 0$, no PSNR degradation is observed for $F = 90$, as this is the base case criteria, where as for $F = 50$, $\Delta PSNR$ is nearly 4 dB. This means that a significant amount of error is incorporated in the image without any approximations to achieve higher compression ratio. We can use this PSNR degradation of $\sigma_{budget}^2 = 0$ for various quality factors, F, as an advantage for achieving power savings with some increased memory. Appropriate F and σ_{budget}^2 are typically provided as inputs based on the application. For example, if $F = 50$ and the acceptable degradation is 4-dB, we cannot approximate any further as 4 dB error is already incorporated due to the quality degradation factor, F. As seen from Figure 6(b), the variance budget for this criteria will be zero and there will be no area and power savings. For $F = 90$, $\Delta PSNR$ budget for 4 dB and the σ_{budget}^2 will be 5,000. The corresponding area and power savings are 12.0% an 16.2%, respectively as shown in Figure 6(a). This implies higher area and power savings are possible using higher F.

Qualitatively, high precision applications should choose higher quality factor, F, with very smaller error budget,

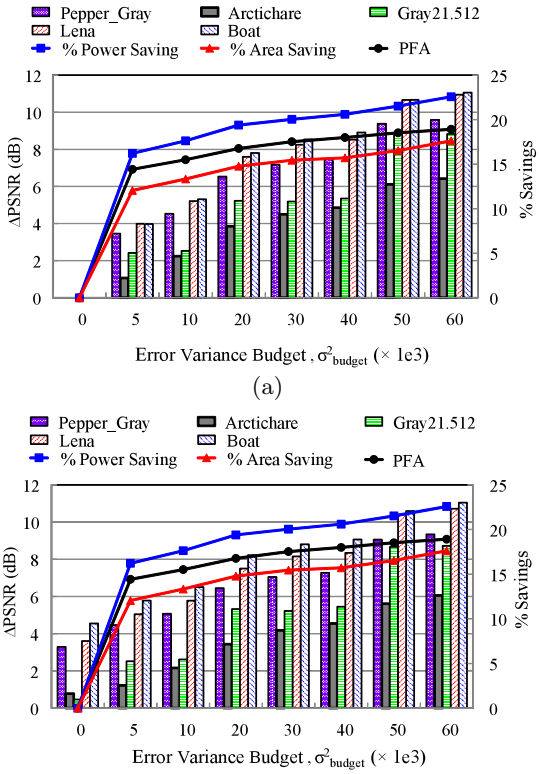

Figure 6: PSNR degradation, $\Delta PSNR$, PFA, area and power savings vs. σ_{budget}^2, for quality factor, (a) $F = 90$ and (b) $F = 50$.

σ_{budget}^2, which will still provide around 12 – 14% savings in both area and power. On the other hand, error tolerant applications should choose higher F with higher error budget and save up to 20% in area and power.

We observe 15% delay improvement for the approximate hardware which can be translated to reduced supply voltage, V'_{dd}. The reduced power, P', for such a voltage-scaled system depends on the global V_{dd}, global clock period, T_c as well as reduced switching capacitance, C'_{sw} as,

$$P' = \frac{C'_{sw} V'^2_{dd}}{2T_c} \tag{18}$$

where $C'_{sw} = C_{sw}(1 - PFA)$, $V'_{dd} = V_{dd}\left(1 - \frac{\Delta T}{T_c}\right)$ and ΔT is the delay change (slack) due to the approximate hardware. From (18), we infer that $\sim 40\%$ power reduction is achievable with voltage scaling with our approximation scheme.

To provide a visual illustration, compressed images with $F = 90$ for four different specified σ_{budget}^2 are shown in Figure 7. The PSNR for the base case, shown in Figure 7(a), is 35.29 dB. Image qualities with three different error variances are shown in the figure. In spite of the significant approximation very little difference in image quality is visible.

The scatter points in Figure 8 represent the maximum pixel error generated for different images incorporating different σ_{budget}^2. For comparison, the relation between σ_{budget}^2 and maximum allowable pixel error, δ_e, shown in (11), is also represented in the figure with a line. It is observed that, the maximum error generally remains below the maximum error budget. The error generated at $\sigma_{budget}^2 = 0$ (without any approximations) is the lossy JPEG compression error, which is not captured in (11).

Finally, we compare the benefit of our approximation scheme,

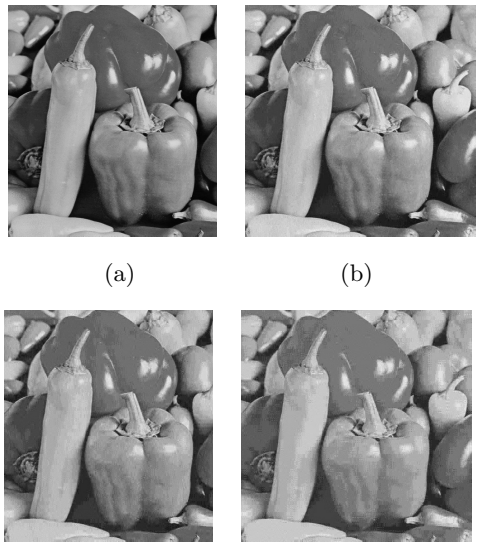

Figure 7: Image quality for peppers_gray ($F = 90$) with (a) σ^2_{budget}=0, PSNR=35.29dB (b) σ^2_{budget}=1.e4, PSNR=30.76dB (c) σ^2_{budget}=3.e4, PSNR=28.11dB (d) σ^2_{budget}=6.e4, PSNR=25.70dB.

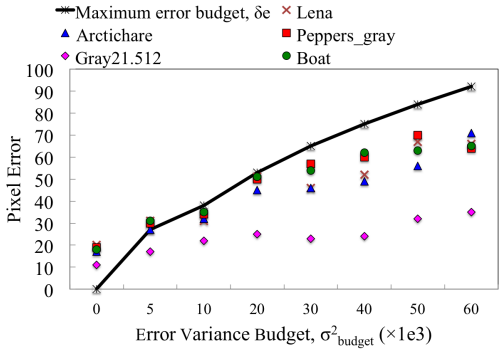

Figure 8: Error budget line, δ_e, and the maximum pixel error for different images for $F = 90$.

where 80 nodes in the 1D DCT are allowed to have independent approximation levels, versus the case where a fixed number of bits are approximated for all nodes in the DCT. The latter case incorporates higher amount of error than approximating variable bit size error generation, i.e., the optimization uses only one variable.

As shown in Figure 6, *PFA* provides a good estimation of area and power savings, and therefore we use this metric to compare the 80-variable and 160-variable optimization with the single-variable optimization in Figure 9. We find that our scheme provides 10% more savings, while the simpler scheme only provides marginal benefits. This is because the sensitivity of many addition and subtraction operations in the DAG is high, and approximation errors are magnified. Our sensitivity-based scheme captures this effect and chooses a lower number of approximate LSBs for high-sensitivity nodes, and more approximate LSBs for low-sensitivity nodes. The result for the 160-variable optimization problem only provides a feasible solution after iterating over 200,000 steps for about three hours. The results of both the 80-variable and 160-variable optimizations provide similar improvements in *PFA*. Increasing the problem space in non-linear optimization leads the solution further away from the optimal point and also increases the CPU time.

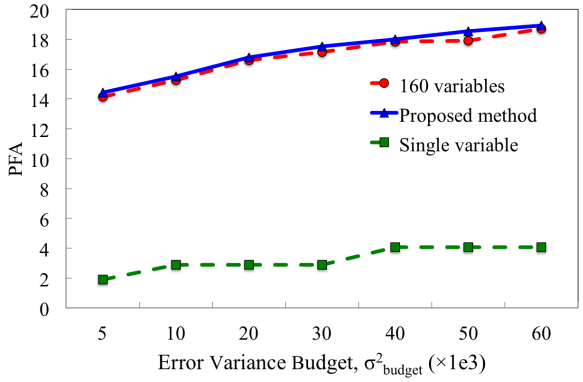

Figure 9: Comparison among proposed method, 160 variables and single variable method.

6. CONCLUSION

We have presented an approach that solves a small integer nonlinear program to optimize the power dissipation of a JPEG compression unit. We formulate the problem using the sensitivity of nodes in the DAG that represents the computation and enable a solution that approximates more (fewer) LSBs at low (high)-sensitivity nodes.

7. REFERENCES

[1] V. Gupta, *et al.*, "Low-Power Digital Signal Processing Using Approximate Adders," *IEEE T. Comput. Aid. D.*, vol. 32, no. 1, pp. 124–137, 2013.

[2] G. Varatkar and N. Shanbhag, "Energy-Efficient Motion Estimation using Error-Tolerance," in *Proc. ISLPED*, pp. 113–118, 2006.

[3] Y. Emre and C. Chakrabarti, "Energy and Quality-Aware Multimedia Signal Processing," *IEEE T. Multimedia*, vol. 15, no. 7, pp. 1579–1593, 2013.

[4] L. N. Chakrapani, *et al.*, "Highly Energy and Performance Efficient Embedded Computing Through Approximately Correct Arithmetic: A Mathematical Foundation and Preliminary Experimental Validation," in *Proc. CASES*, pp. 187–196, 2008.

[5] J. Miao, *et al.*, "Modeling and Synthesis of Quality-Energy Optimal Approximate Adders," in *Proc. ICCAD*, pp. 728–735, 2012.

[6] W. Chen, *et al.*, "A Fast Computational Algorithm for the Discrete Cosine Transform," *IEEE T. Commun.*, vol. 25, no. 9, pp. 1004–1009, 1977.

[7] B. Lee, "A New Algorithm to Compute the Discrete Cosine Transform," *IEEE T. Acoust. Speech*, vol. 32, no. 6, pp. 1243–12457, 1984.

[8] C. Loeffler, *et al.*, "Practical Fast 1-D DCT Algorithms with 11 Multiplications," in *Proc. ICASSP*, vol. 2, pp. 988–991, 1989.

[9] A. Mammeri, *et al.*, "Modeling and Adapting JPEG to the Energy Requirements of VSN," in *Proc. ICCCN*, pp. 1–6, 2008.

[10] J. Park, *et al.*, "Dynamic Bit-Width Adaptation in DCT: An Approach to Trade Off Image Quality and Computation Energy," *IEEE T. VLSI Syst*, vol. 18, no. 5, pp. 787–793, 2010.

[11] K. Nepal, *et al.*, "ABACUS: A Technique for Automated Behavioral Synthesis of Approximate Computing Circuits," in *Proc. DATE*, pp. 361:1–361:6, 2014.

[12] C. Li, *et al.*, "Joint Precision Optimization and High Level Synthesis for Approximate Computing," in *Proc. DAC*, pp. 1–6, 2015.

[13] "JPEG — Wikipedia, the free encyclopedia." https://en.wikipedia.org/wiki/JPEG [Online; accessed 04-April-2016].

[14] "Standard Image Database." http://sipi.usc.edu/database/.

[15] "KNITRO User Manual." http://www.ziena.com/docs/Knitro90_UserManual.pdf.