A Methodology for Energy-Quality Tradeoffs Using Imprecise Hardware

Jiawei Huang
Computer Engineering
University of Virginia
jh3wn@virginia.edu

John Lach
Electrical and Computer Engineering
University of Virginia
jlach@virginia.edu

Gabriel Robins
Computer Science
University of Virginia
robins@cs.virginia.edu

ABSTRACT

Recent studies have demonstrated the potential for reducing energy consumption in integrated circuits by allowing errors during computation. While most proposed techniques for achieving this rely on voltage overscaling (VOS), this paper shows that *Imprecise Hardware (IHW)* with design-time structural parameters can achieve orthogonal energy-quality tradeoffs. Two IHW adders are improved and two IHW multipliers are introduced in this paper. In addition, a simulation-free error estimation technique is proposed to rapidly and accurately estimate the impact of IHW on output quality. Finally, a quality-aware energy minimization methodology is presented. To validate this methodology, experiments are conducted on two computational kernels: DOT-PRODUCT and L2-NORM – used in three applications – Leukocyte Tracker, SVM classification and K-means clustering. Results show that the Hellinger distance between estimated and simulated error distribution is within 0.05 and that the methodology enables designers to explore energy-quality tradeoffs with significant reduction in simulation complexity.

Categories and Subject Descriptors

G.1.6 [**Numerical Analysis**]: Constrained optimization

General Terms

Algorithms

Keywords

Imprecise hardware, energy-quality tradeoff, static error estimation

1. INTRODUCTION

High power consumption is one of the greatest challenges currently facing IC designers. Although circuit-level techniques such as dynamic voltage and frequency scaling (DVFS), as well as sub- and near-threshold operation have proved effective in power reduction, they are fundamentally limited by the critical path of the circuit. Recently, a new design philosophy has emerged that relaxes the absolute correctness requirement to achieve further power reductions. For example, application noise tolerance [1] combines a voltage-overscaled computation core with a low-precision error-compensation core. Significance driven computation [2] identifies functionally non-critical parts of an algorithm and employs VOS to save power. Both techniques exploit the error-tolerant nature of the algorithms being implemented and use V_{dd} as the leverage to tradeoff quality for power. However, a good understanding of the algorithm is usually required to identify functionally non-critical components that could be "imprecisely" implemented without excessively degrading the output quality. In addition, the system must be *simulated* under a range of V_{dd} in order to find the optimal power-quality tradeoff, which is typically a time-consuming process.

This paper presents a generalized methodology for energy [1]-quality tradeoffs with two unique features. First, it incorporates "variables" for imprecise computation other than V_{dd}; namely RTL structural parameters for deterministic design-time energy-quality tradeoffs. The specific IHW components introduced here are parameterized ALUs. IHW is orthogonal to existing V_{dd}-lowering techniques, as VOS can be applied on top of IHW to achieve even higher energy reduction. Second, the methodology utilizes a novel static error estimation method that models the output error distribution based on the input distribution and design parameters. This method enables the automated exploration of the energy-quality space without computational-intensive simulations at each design point. It is also general enough to be used by VOS designs for rapid quality evaluation to speed up V_{dd} selection.
Table 1 lists two kernel functions common in multimedia, recognition and mining applications [3] and three examples of such applications. These kernels and applications will be used to demonstrate and validate the proposed methodology. In principle, this methodology can be used to explore energy-quality tradeoffs in any error-resilient applications with computational kernels that can be implemented with IHW.

Table 1. % application runtime spent in computation kernels

Kernel	Application	Runtime%
$DOT-PRODUCT(X,Y)=\sum_{i=1}^{n} x_i * y_i$	Leukocyte Tracker	22%
$L2-NORM(X,Y)=\sum_{i=1}^{n}(x_i - y_i)^2$	SVM	98%
	K-means	49%

Major contributions of this work include:
- a generalized quality-aware energy minimization methodology,
- a fast and accurate static error estimation method, and
- design of imprecise multipliers based on imprecise adders.

The rest of the paper is organized as follows. Section 2 reviews related work in this area and highlights the motivation of this work. Section 3 introduces two existing imprecise adders as well as some improvements and adaptations to use them to build imprecise multipliers. Section 4 introduces the static error estimation method. The quality-aware energy-minimization methodology is described in Section 5, followed by application-

[1] This paper will focus on *energy per operation (E/op)* instead of power, but the methodology is applicable to any hardware metric such as power, area, energy-delay product, etc.

level energy-quality tradeoff results in Section 6. Section 7 concludes the paper.

2. BACKGROUND AND RELATED WORK

Most of the prior work on imprecise computation focuses on power reduction through VOS [1, 2]. Since traditional circuits are designed such that most paths have delays close to those of critical paths, naive VOS will likely induce massive timing violations and circuit failure. These techniques attempt to either correct errors with redundant circuits [1] or delay the onset of massive errors through timing path rebalancing [4]. Mohapatra et. al. [3] suggest a way to memorize the timing errors in a counter and make corrections over longer time intervals. However, these techniques do not fundamentally change the circuit structure to enable energy-quality tradeoffs at design time. Another problem is finding the optimal V_{dd}. There is no easy way to predict the output quality at a certain V_{dd} level except through time-consuming detailed circuit simulation.

The correctness requirement in error-tolerant applications can be further relaxed, leaving errors *uncorrected*. For example, users are unlikely to notice small/rare degradations in multimedia quality, and computation errors often do not affect the results of recognition or data mining analyses. Therefore, many high-energy circuit structures could be simplified (such as breaking long adder carry propagation chains with constant 1s or 0s) with a tolerable impact on application-level quality. Such design-time techniques could be used in conjunction with runtime techniques (e.g., VOS) to achieve more desirable energy-quality tradeoffs.

Since IHW inevitably leads to some loss of accuracy, it is particularly important to be able to evaluate its effect on output quality. The static error estimation technique proposed in Section 4 achieves this goal by leveraging statistical analysis to propagate the error distribution through a system of arithmetic operators. Although the application-level quality impact still needs to be evaluated through simulation, arithmetic kernel-level quality estimation can significantly reduce the number of design points that need to be simulated. Most suboptimal design points are eliminated at the kernel level by the static error estimator. With the exception of the initial simulation to characterize IHW components, no simulation is required at the kernel-level, and the same characterization data can be reused for arbitrary input distributions.

3. IMPRECISE ADDERS AND MULTIPLIERS

Adders and multipliers are used extensively in multimedia and data mining applications. Imprecise implementations of adders and multipliers have the most direct impact on system energy and output quality. This section presents two imprecise adder designs in the literature, and introduces new imprecise multiplier designs.

3.1 ACA Adder

The Almost Correct Adder (ACA) [5] is a modified version of the traditional Kogge-Stone adder (KSA). ACA leverages the fact that under random inputs, the vast majority of the actual timing paths are much shorter than the worst-case critical path. Table 2 gives the probability of two random 64-bit inputs triggering a critical path longer than K. Even with K much smaller than 64, the probability of critical path violation is quite small and that probability decreases rapidly with larger K. ACA then uses a tree structure to compute the *propagate* and *generate* signals similar to KSA but assumes the longest run of *propagate* never exceeds K, i.e., Sum_i is computed using only $A_i \cdots A_{i-K+1}$ and $B_i \cdots B_{i-K+1}$. Its worst case delay is $\log_2(K)$. ACA's structure is essentially a trimmed KSA tree. A smaller tree translates to lower delay, smaller area and less energy per addition.

Table 2. Prob. of a random propagate chain exceeding *K* bits

K	12	16	24	30
Probability	0.0024	1.22×10^{-4}	2.4×10^{-7}	9.1×10^{-10}

Errors occur in ACA when the inputs trigger a *propagate* chain longer than K. For example, when A and B are exactly complementary, the *propagate* chain will extend the full adder's length. To produce the correct Sum_i, all the bits from both inputs will be needed, but ACA speculates and approximates it with the *propagate* chain from bit i down to $i-K+1$ with the carry-in set to constant 0. In case of incorrect speculation, a large error will appear in Sum_i. The largest error occurs when bit i is the MSB. Errors with such characteristics are called **infrequent large-magnitude (ILM)** errors [6]: they occur rarely, but whenever they do, their magnitude tends to be large. Energy-quality tradeoffs can be achieved by tuning the design parameter K.

3.2 ETAIIM Adder

The Modified Error-Tolerant Adder Type II (ETAIIM) [7] is another type of imprecise adder based on the Ripple-carry adder (RCA). RCA has a simple linear *propagate* chain. ETAIIM works by partitioning the *propagate* chain into segments of variable widths. The carry bits across two segments are truncated to zero. In order to provide higher precision for higher-order bits, segments are wider (i.e., contain more bits) on the MSB side than on the LSB side. ETAIIM has two parameters: *BPB* (bits per block) and L (the number of blocks used for generating the MSB). A block refers to the smallest segment, which is usually located at the LSB. The maximum error magnitude of ETAIIM is limited by $BPB \times L$. However, carry generation across blocks is common; therefore, errors occur quite frequently in ETAIIM. These errors are called **frequent small-magnitude (FSM)** errors [6] because their magnitudes are bounded by the design parameters and are usually small compared to ILM errors.

3.3 Improving Imprecise Adders

The original ACA and ETAIIM designs do exhibit a weakness. For simplicity, both designs use a constant 0 as the carry-in at the cut-off point of the critical path, but this leads to negatively-biased errors because 0 is an underestimation of the carry-in bit. Similarly, constant 1 will produce positively-biased errors. One possible improvement is to take the carry-in from the bit immediately before the *propagate* chain. For ACA, this means the propagate chain formed by $A_i \cdots A_{i-K+1}$ and $B_i \cdots B_{i-K+1}$ will take A_{i-K} (or B_{i-K}) as the carry-in. For ETAIIM, it means the carry bit across blocks will be taken from the highest bit in the previous block. If the inputs are random during the computation, every bit has a 50% probability of being 0 or 1. This will eventually produce an unbiased error distribution in the sum. Table 3 is obtained from simulating the summation of 20 numbers randomly drawn from [-0.5, 0.5] using ETAIIM adder (*BPB*=8, *L*=4). The anti-biasing technique notably improves statistical error metrics.

Table 3. Error metrics improvement with anti-biasing

Metrics	Original	w. Anti-biasing
Error Rate	12.3%	6.9%
Mean Error Magnitude	6.3×10^{-8}	3.3×10^{-8}

3.4 Imprecise Multipliers

Despite the lack of imprecise multipliers in the literature, it is possible to build imprecise multipliers based on imprecise adders. A typical multiplier consists of three stages: partial product generation, partial product accumulation and a final stage adder [8]. The idea of building an imprecise multiplier is simple: replace the final stage adder with an imprecise adder. The ACA and ETAIIM adders will thus yield corresponding ACA and ETAIIM multipliers. For the other two stages, we adopt the popular simple partial product generation (shifted versions of the multiplicand without recoding) [8] and Wallace-tree partial product accumulator (3:2 compressor tree) [9]. These choices will influence the actual energy numbers but they do not affect the ability to perform energy-quality tradeoffs.

Table 4 compares the energy-delay product (EDP) of various precise and imprecise adders and multipliers. They are all synthesized to their respective critical path delays in 130nm technology, and imprecise ALUs are operated at lower voltages to match the speed of their precise counterparts. As seen from the table, imprecise ALUs consume significantly less E/op than their precise counterparts at the same delay due to their simplified logic structures.

Table 4. E/op and area of precise and imprecise ALUs

ALU	E/op (pJ)	Delay (ns)	EDP (pJ·ns)
KSA64	8.47	0.8	6.776
ACA64 (K=16)	4.96		3.968
RCA64	5.48	5.3	29.044
METAII (BPB=4, L=4)	0.527		2.793
MULT64_KSA	413.18	2.6	1074.268
MULT64_ACA (K=32)	365.98		951.548
MULT64_RCA	174.8	11.1	1940.28
MULT64_METAII (BPB=4, L=4)	82.56		916.416

4. STATIC ERROR ESTIMATION

While many CAD tools exist to evaluate the energy consumption of an integrated circuit, quality evaluation capability is far less common – i.e., determining how much the imprecise output differs from the precise output. In all VOS techniques, quality is evaluated by running Monte Carlo simulations, since the relationship between the circuit variables and the output cannot be easily derived. There is a fundamental drawback to this approach: the simulation time grows exponentially with data width and computation length. For example, a length-10 DOT-PRODUCT with 32-bit numbers would require $32^{20} \approx 1.3 \times 10^{30}$ different input vectors to cover the entire input space.

This section presents a static error estimation technique that eliminates the need for simulation during quality evaluation at the kernel level. We make two assumptions here: 1) the only operations involved are additions and multiplications, and 2) input data (X and Y) are independent. Assumption 1 is satisfied in both kernel functions in Table 1 and in many error-tolerant application domains. Assumption 2 is necessary to prevent, for example, the product $X * Y$ reducing to certain forms of X^2. If a squaring operation is treated as a normal two-operand multiplication, the estimation accuracy will be significantly lower. In the DOT-PRODUCT kernel, the probability of any X_i=Y_i is quite low so this assumption is usually satisfied. Estimation of the squaring operation in L2-NORM will be discussed in Section 4.3. All the adders and multipliers in this discussion will be 64-bit wide. The number representation is 2's complement 4_60, with 4 bits (including sign bit) before the decimal point and 60 bits after. In multiplication, the product format is 8_120. All input data are scaled to prevent overflow during computation.

4.1 Probability Mass Function (PMF)

Probability Mass Function (PMF) is a way of representing the statistical distribution of any discrete data/error. It can be visualized as a bar chart on the magnitude vs. frequency plane as shown in Figure 1.

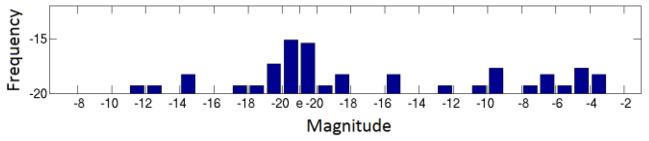


Figure 1. PMF examples

Each bar indicates a non-zero data probability. The location of a bar on the x-axis indicates the magnitude range of the data and the height indicates its frequency of occurrence. The taller a bar is, the more frequent the data occur. Both the x-axis and y-axis are logarithmic-scaled in order to cover a wider frequency-magnitude range. For example, a bar bounded by marker -8 and -7 with a height -10 means that the probability of observing the data between magnitude 2^{-8} and 2^{-7} is 2^{-10}. The e symbol in the middle of the x-axis represents zero; thus, bars to the left have negative magnitude and those to the right have positive magnitude. The sum of the heights of all the bars in a PMF is equal to the probability of the data being non-zero (P_{NZ});. The probability of zero is therefore implicitly obtained by $1-P_{NZ}$. When PMF is used to represent an error distribution, it is possible that $P_{NZ} < 1$. In this case P_{NZ} represents the total error probability P_e and $1-P_e$ gives the error-free probability. Within each bar, the data is assumed to be uniformly distributed.

4.2 Modified Interval Arithmetic (MIA)

Interval Arithmetic (IA) [10] is a classical method to estimate variable ranges during numerical computations. It uses a single interval $[x_l, x_r]$ to represent each variable. When the variable takes part in computation, its interval goes through corresponding IA to produce the output interval. Provided that data are not correlated, the bounds given by IA are tight. However, in many cases data and error distributions such as in Figure 1 cannot be represented by a single uniform distribution.

Modified Interval Arithmetic (MIA) [11] extends IA by using *multiple* intervals to represent a distribution to enhance accuracy. MIA can be easily mapped to PMF: each PMF bar corresponds to one interval in MIA. The entire MIA can be formalized as

$$MIA(x) = P(a \le x \le b), \text{if } a \le x \le b$$

When an error distribution is represented in MIA, the total error probability is given by $\int MIA(x)$.

When two intervals operate with each other, their resulting interval observes the following rules:

$$[x_{l1}, x_{r1}] + [x_{l2}, x_{r2}] = [x_{l1} + x_{l2}, x_{r1} + x_{r2}]$$
$$[x_{l1}, x_{r1}] * [x_{l2}, x_{r2}] = [\min(x_{l1}x_{l2}, x_{l1}x_{r2}, x_{r1}x_{l2}, x_{r1}x_{r2}), \max(x_{l1}x_{l2}, x_{l1}x_{r2}, x_{r1}x_{l2}, x_{r1}x_{r2})]$$

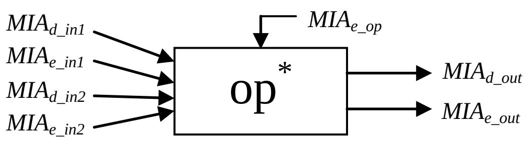
For operations between two MIAs, each IA from the first MIA must perform that operation with each IA from the second MIA and the resulting IAs are merged into a single MIA. While merging, IAs of the same intervals are combined into one IA with its probability being equal to the sum of the constituent IA probabilities.

4.3 Propagating MIA across IHW

Rules in the previous subsection assume precise operation. They must be modified to account for imprecise operators.

The first step is to use a common data structure (MIA_d, MIA_e) to represent any data during imprecise operation. MIA_d is the error-free MIA obtained assuming all operators are precise, while MIA_e is the pure error MIA introduced by imprecise operators. The sum of MIA_d and MIA_e gives the actual data MIA.

It then becomes necessary to build a model to obtain the output (MIA_{d_out}, MIA_{e_out}) from input (MIA_{d_in}, MIA_{e_in}). The imprecise operator (marked with *) will also introduce MIA_{e_op}, which can be regarded as additive noise to the system. We have derived the relationships between these quantities (Figure 2).



$$\underline{\textbf{ADD}}: MIA_{d\_out} = MIA_{d\_in1} + MIA_{d\_in2}$$
$$MIA_{e\_out} = MIA_{e\_in1} + MIA_{e\_in2} + MIA_{e\_add}$$
$$\underline{\textbf{MUL}}: MIA_{d\_out} = MIA_{d\_in1} * MIA_{d\_in2}$$
$$MIA_{e\_out} = MIA_{d\_in1} * MIA_{e\_in2} + MIA_{d\_in2} * MIA_{e\_in1} + MIA_{e\_in1} * MIA_{e\_in2} + MIA_{e\_mul}$$
$$\underline{\textbf{SQUARE}}: MIA_{d\_out} = MIA_{d\_in}^2$$
$$MIA_{e\_out} = MIA_{e\_in}^2 + 2 * MIA_{d\_in} * MIA_{e\_in} + MIA_{e\_square}$$

Figure 2. MIA propagation rules for ADD/MUL/SQUARE

Operations between MIAs follow the rules given in Section 4.2. Notice that SQUARE is separated from MUL because it cannot be obtained from simple MIA operations such as * and +. Even if X and Y have the same distribution, the distribution of $X*Y$ and X^2 will be different. The modeling of SQUARE will rely on characterization.

MIA_{e_add}, MIA_{e_mul} and MIA_{e_square} are attributes of the operator determined by the circuit design parameters. They can be obtained by simulation. The process of obtaining MIA_{e_op} through simulation is called *characterization of IHW*. To characterize ADD and MUL, we randomly draw data from single bars from both inputs' MIAs (i.e., draw first operand from $[2^i, 2^{i+1}]$ and draw second operand from $[2^j, 2^{j+1}]$) and perform the imprecise operation. Simulation is made possible by creating a functional model of the imprecise adders and multipliers written in C. The resulting MIA_d and MIA_e are then stored into a matrix at index (i, j). When the entire matrix is populated, we can later use it to quickly retrieve MIA_{e_op} during MIA propagation. For the unary operator SQUARE, the result is stored in a vector instead of a matrix and we need two vectors for SQUARE: one for looking up errors (MIA_{e_square}), and the other for looking up squared data ($MIA^2_{d\_in}$ and $MIA^2_{e\_in}$). IHW can be characterized *a priori* and each IHW configuration (i.e., a unique setting of *BPB, L* and *K*) needs to be characterized only once. The characterization data can then be reused many times for different kernel input workloads.

In summary, kernel-level MIA propagation follows three steps:

1) Construct the characterization vector/matrix by simulating the IHW with inputs being drawn from various $[\pm 2^i, \pm 2^{i+1}]$ intervals.
2) During propagation, use the input MIAs to look up the characterization vector/matrix to obtain MIA_{e_op}.
3) Apply rules in Figure 2 to obtain output MIA.

Step 2 and 3 may need to be repeated because the output MIA normally becomes the input MIA of the next round of computation. The final MIA_d and MIA_e accurately describe the data and error distribution of the kernel output and they can be used to evaluate output quality. Common quality metrics such as error rate and mean error magnitude are computed as follows:

$$\textit{error rate} = \int MIA_e(x)$$
$$\textit{mean error magnitude} = \int |x \cdot MIA_e(x)|$$

Static MIA propagation is much faster than Monte Carlo simulation because no actual computation is performed. It is the distributions (in the form of MIA) rather than actual data that are being propagated.

4.4 Experimental Results

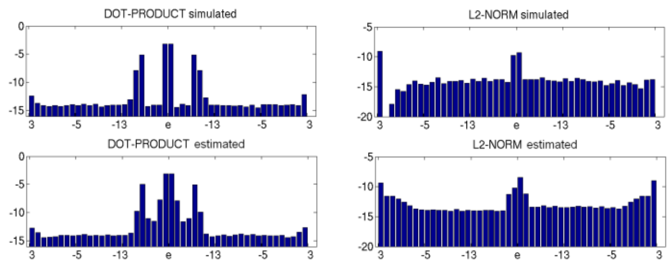


Figure 3. Error MIAs of DOT-PRODUCT and L2-NORM

Figure 3 shows the final error MIAs after performing a size-25 DOT-PRODUCT and a size-49 L2-NORM using both Monte Carlo simulation and static estimation. DOT-PRODUCT contains an ACA adder with K=16 and an ETAIIM multiplier with BPB=8 and L=4; L2-NORM contains an ACA adder with K=16 and an ACA multiplier with K=24. Table 5 compares the speed and accuracy of simulated and estimated error MIAs. All experiments are run on a dual-core Xeon 2.4GHz with 32GB memory. The simulation size is 500,000 and is regarded as the ground truth. As seen in the table, the speed improvement is dramatic and the simulated and estimated error distributions are very close. For example, a Hellinger distance[2] of 0.05 is comparable to 1 million random samples from two uniform distributions between [-1, 1] generated by Matlab's default Mersenne Twister algorithm [13].

Table 5. Speed and accuracy comparison between simulation and static estimation

Kernel	Sim. time	Est. time	Hellinger distance
DOT-PRODUCT	565 hr	13 s	0.05
L2-NORM	620 hr	6 s	0.02

5. QUALITY-AWARE ENERGY MINIMIZATION FLOW

The energy-quality optimization problem can be formulated in many different ways, such as $\textit{energy}^a/\textit{quality}^b$ cost minimization or quality maximization subject to an energy constraint. This paper focuses on solving the quality-constrained energy minimization problem:

```
minimize:      E(x₀, x₁, . . . , xₙ)
subject to:    Q(x₀, x₁, . . . , xₙ) >= Q₀
```

[2] Statistical measure of similarity between two distributions – smaller values indicate higher similarity [12].

where E denotes the energy consumed while performing a kernel computation; Q denotes the resultant quality. $x_0, x_1, \ldots, x_n$ are circuit structural parameters such as *BPB*, *L* and *K*. Assuming the adders and multipliers are restricted to 64-bit, then the x vector for the DOT-PRODUCT kernel is as follows:

$$[add_{mode}\ BPB_{add}\ L_{add}\ K_{add}\ mul_{mode}\ BPB_{mul}\ L_{mul}\ K_{mul}]$$

add_{mode}/mul_{mode} is an integer representing IHW type: 0=KSA, 1=ACA, 2=ETAIIM, 3=RCA. L2-NORM needs four additional parameters for its subtractor. Circuit operating conditions such as V_{dd} and frequency can also be included in the x vector, and it is part of the ongoing work of combining IHW with VOS. There are certain restrictions on each parameter, such as the requirement that the adder width (64) must be divisible by *BPB* and $BPB \times L$ cannot exceed 64. Parameters will be swept in their valid ranges only.

Including precise (KSA/ACA) designs, there are a total of 39 adder designs and 101 multiplier designs. DOT-PRODUCT needs 1 adder and 1 multiplier, forming a space of 8 variables and 3939 design points. L2-NORM needs 2 adders and 1 multiplier, forming a space of 12 variables and 154,000 points.

Since all the parameters must be integers, this is an integer programming problem. Matlab offers a genetic algorithm function (GA) to solve these types of problems. It requires two routines to calculate E and Q respectively. For energy calculation, parameterized RTL models were developed for ACA/ETAIIM adders and multipliers and the RTL for KSA/RCA was obtained online [14]. We then synthesized the models into netlists using Cadence RC Compiler in ST 130nm CMOS technology and simulated 1000 random additions and 100 random multiplications using Cadence Ultrasim. Energy per operation can be extracted from the simulation waveforms. An energy model is subsequently built using curve-fitting to extrapolate to the entire parameter space. For simplicity, the energy consumed in the control logic is ignored and the sum of ALU energies is used to represent the energy of the kernel.

For calculation of quality, MIA propagation was implemented in C++ as an extension to the *libaffa* project [15]. The workload is written into a text file with each line in the following format:

```
MUL ETAIIM 8 4 0 4 60 -1 1 -1 1
```

This specifies the operator's parameters (ETAIIM multiplier with *BPB*=8, *L*=4), input format (4_60), and input data ranges ([-1, 1]). A program parses this file and the characterization vector/matrix files, performs the MIA propagation, and writes the output data and error MIA into a result file. A final Matlab script extracts error rate and mean error magnitude metrics from the result file.

5.1 Experimental Results

The methodology was tested on two kernels: size-8 DOT-PRODUCT with inputs in [-1, 1] and size-10 L2-NORM with inputs in [-0.25, 0.25]. Their sizes and dynamic ranges are based on the actual computation and data range profiled while running their corresponding applications. Two quality metrics are evaluated: error rate and mean error magnitude. By setting the quality constraint at different values between [2^{-10}, 2^{-1}], the optimizer is able to produce the energy-quality tradeoff curves in Figure 4. As a comparison, we also show curves obtained by running an exhaustive search on all possible design points. In all four figures, the optimizer curves follow the exhaustive-search curves with a maximum deviation of 2%. Both kernels enjoy a region of about 10% energy reduction with graceful quality degradation. All the curves are significantly lower than the lowest energy achievable by precise designs (136.44pJ for DOT-PRODUCT and 140.4pJ for L2-NORM).

6. APPLICATION-LEVEL ANALYSIS

Since the application-level quality can only be obtained through simulation, it is difficult to extend the previous methodology to the application level. Simulating the application with IHW is usually 2-3 orders of magnitude slower than with precise hardware, because the host machine cannot use a single ALU instruction to perform an imprecise operation. However, kernel-level solutions can facilitate the application-level exploration process. The first step is to solve the kernel-level problem multiple times using static analysis, each time with a different quality constraint value. Then, assuming application-level quality is a monotonic function of kernel-level quality, the application can be simulated using only the points identified during the kernel-level exploration. The same genetic algorithm (GA) can then be applied to obtain the minimum-energy point given an application-level quality requirement. This section presents experimental results at the application level assisted by kernel-level exploration. The goal of these experiments is to demonstrate the energy-quality behavior of different applications under IHW implementation and the benefits of the proposed methodology.

The three applications chosen to evaluate the proposed methodology are shown in Table 1. Leukocyte Tracker implements an object-tracking algorithm [16] in which an important step is to compute the sum of gradients on the 8 neighboring pixels. SVM is a classification algorithm that consists of a training stage and a prediction stage. The training stage involves computing the Euclidean distance of two data points (called radial basis function) in order to map them into a higher dimensional space. K-means is a data clustering algorithm; the basic operation is calculating the distance between two data points. The Euclidian distance is commonly used. Both K-means and SVM use the L2-NORM kernel, whereas Leukocyte Tracker uses the DOT-PRODUCT kernel. In each application, the corresponding kernel represents a significant percentage of the runtime (Table 1). The source code for Leukocyte and K-means is obtained from the Rodinia benchmark suite [17] and SVM from libSVM [18]. All benchmarks provide sample input data. In Leukocyte tracker we tracked 36 cells in 5 frames; in SVM we attempted to classify 683 breast cancer data points with 10 features into 2 classes; in K-means, we tried to cluster 100 data points with 34 features into 5 clusters.

Quality metrics for the three applications are defined as follows. For Leukocyte, the center locations of the tracked cells are compared with the locations returned by the precise implementation. The *average cell-center deviation* serves as a good negative quality metric. *Classification accuracy* is a well established quality metric for SVM. Finally, for K-means, *mean centroid distance* [3] is used.

Before simulation, the programs are first profiled to determine the dynamic range of data during kernel computation. If the dynamic range is greater than the characterized data range, it is necessary to perform scaling on the input and output data. Certain applications, such as SVM and Leukocyte, already incorporate data normalization into their algorithm so no scaling is necessary.

The design points returned during kernel-level optimization are then used to rewrite the kernel portions of the three applications using those imprecise designs.

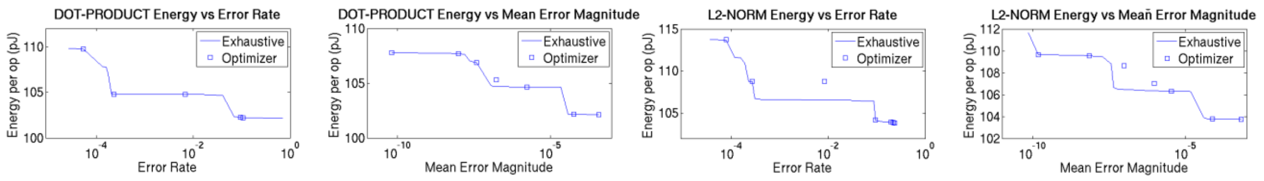


Figure 4. Kernel-level energy-quality tradeoffs

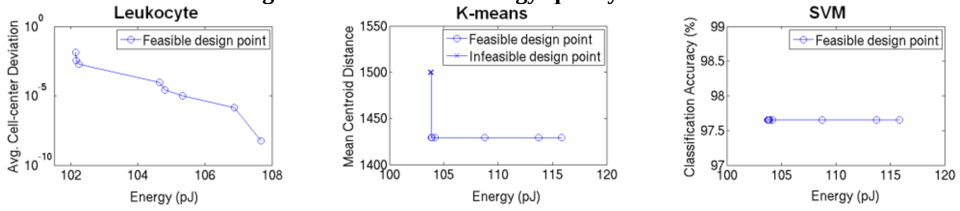


Figure 5. Application-level energy-quality tradeoffs

The final application-level energy-quality tradeoff curves are shown in Figure 5. Since running a SPICE simulation of the entire application to obtain its energy is prohibitively slow, the kernel's energy was used to represent the entire application's energy. Among the three applications, Leukocyte has a smooth quality-energy transition region. At its lowest-energy point (102.24pJ), the mean deviation from precise outputs is merely 0.1 pixels. Its energy is 25% lower than the 136.44pJ precise design. For K-means, the mean centroid distance remains unchanged (1429.22) above the 103.8pJ energy point (i.e., a 26% reduction over precise design). Any design below that energy point failed to converge during simulation. A similar situation is observed in SVM where the critical energy point is 103.76pJ.

Table 6. Number of designs points simulated

Search method	Leukocyte Tracker	SVM	K-means
Exhaustive search	3,939	153,621	153,621
GA (app-level)	887	1,343	1,343
Proposed methodology	**15**	**17**	**17**

Table 6 compares the number of design points that needed to be simulated in order to generate the application-level energy-quality tradeoff curves in Figure 5. Exhaustive search simulates all the design points once, while applying GA at the application-level simulates only a subset. The proposed methodology simulates the least number of design points because it only chooses those points on the optimal kernel-level energy-quality curves.

7. CONCLUSIONS AND FUTURE WORK

This paper presents a methodology to find the lowest-energy design for certain computation kernels given a quality-constraint. This methodology leverages IHW with design-time structural parameters to achieve energy-quality- tradeoffs. It requires no simulation at the kernel level, and the simulation effort at the application level is significantly reduced. Experiments show that the methodology can produce results close to exhaustive search and the runtime is orders-of-magnitude shorter than Monte Carlo simulation. Extending this methodology to support VOS and peak error bounding estimation are valuable future research projects.

8. ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation, under grants IIS-0612049 and CNS-0831426.

9. REFERENCES

[1] Shim, B., Sridhara, S., Shanbhag, N. 2004, Reliable low-power digital signal processing via reduced precision redundancy, *IEEE Transactions on VLSI Systems*, 12(5):497-510.

[2] Mohapatra, D., Karakonstantis, G., Roy, K. 2009, Significance driven computation: a voltage-scalable, variation-aware, quality-tuning motion estimator, *ISLPED*, pp. 195-200.

[3] Mohapatra, D., Chippa, V.K., Raghunathan, A., Roy, K. 2011, Design of voltage-scalable meta-functions for approximate computing, *DATE*, pp. 1-6.

[4] Kahng, A., Kang, S., Kumar, R., Sartori, J. 2010, Slack redistribution for graceful degradation under voltage overscaling, *ASP-DAC*, pp. 825-831.

[5] Verma, A.K., Brisk, P., Ienne, P. 2008, Variable latency speculative addition: A new paradigm for arithmetic circuit design, *DATE*, pp. 1250-1255.

[6] Huang, J., Lach, J. 2011, Exploring the fidelity-efficiency design space using imprecise arithmetic, *ASP-DAC*, pp.579-584.

[7] Zhu, N., Goh, W.L., Yeo, K.S. 2009, An enhanced low-power high-speed adder for error tolerant application, *ISIC*, pp. 69-72.

[8] Ercegovac, M.D., Lang, T. 2004, *Digital Arithmetic*. Morgan Kaufmann Publishers.

[9] Wallace, C.S. 1964, A suggestion for fast multipliers. *IEEE Trans. Electron. Comput.* EC-13(1):14-17.

[10] Moore, R.E. 1966, *Interval Analysis*, Prentice-Hall.

[11] Huang, J., Lach, J., Robins G. 2011, Analytic error modeling for imprecise arithmetic circuits, *SELSE*.

[12] Nikulin, M.S. 2001, Hellinger distance, *Encyclopaedia of Mathematics*, Springer, ISBN 978-1556080104.

[13] Matsumoto, M., Nishimura, T. 1998, Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator, *ACM Transactions on Modeling and Computer Simulation*, 8(1):3-30.

[14] http://www.aoki.ecei.tohoku.ac.jp/arith/mg/index.html

[15] http://savannah.nongnu.org/projects/libaffa

[16] Ray, N., Acton, S.T. 2004, Motion gradient vector flow: an external force for tracking rolling leukocytes with shape and size constrained active contours, *IEEE Transactions on Medical Imaging*, 23(12):1466-1478.

[17] Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J.W., Lee, S.-H., Skadron, K. 2009, Rodinia: A benchmark suite for heterogeneous computing, *IISWC*, pp. 44-54.

[18] Chang, C., Lin, C. 2011, LIBSVM: a library for support vector machines, *ACM Transactions on Intelligent Systems and Technology*, 2(1)27:1-27:27.