Quality Configurable Reduce-and-Rank for Energy Efficient Approximate Computing

Arnab Raha, Swagath Venkataramani, Vijay Raghunathan, and Anand Raghunathan
School of Electrical and Computer Engineering, Purdue University
{araha,venkata0,vr,raghunathan}@purdue.edu

***Abstract*—Approximate computing is an emerging design paradigm that exploits the intrinsic ability of applications to produce acceptable outputs even when their computations are executed approximately. In this work, we explore approximate computing for a key computation pattern, Reduce-and-Rank (RnR), which is prevalent in a wide range of workloads including video processing, recognition, search and data mining. An RnR kernel performs a reduction operation (*e.g.*, distance computation, dot product, L1-norm) between an input vector and each of a set of reference vectors, and ranks the reduction outputs to select the top reference vectors for the current input. We propose two complementary approximation strategies for the RnR computation pattern. The first is *interleaved reduction-and-ranking*, wherein the vector reductions are decomposed into multiple partial reductions and interleaved with the rank computation. Leveraging this transformation, we propose the use of intermediate reduction results and ranks to identify future computations that are likely to have low impact on the output, and can hence be approximated. The second strategy, *input similarity based approximation*, exploits the spatial or temporal correlation of inputs (*e.g.*, pixels of an image or frames of a video) to identify computations that are amenable to approximation. These strategies address a key challenge in approximate computing – identification of which computations to approximate – and may be used to drive any approximation mechanism such as computation skipping and precision scaling to realize performance or energy improvements.**

A second key challenge in approximate computing is that the extent to which computations can be approximated varies significantly from application to application, and across inputs for even a single application. Hence, quality configurability, or the ability to automatically modulate the degree of approximation at runtime is essential. To enable quality configurability in RnR kernels, we propose a kernel-level quality metric that correlates well to application-level quality, and identify key parameters that can be used to tune the proposed approximation strategies dynamically. We develop a runtime framework that modulates the identified parameters during execution of RnR kernels to minimize their energy while meeting a given target quality. To evaluate the proposed concepts, we designed quality-configurable hardware implementations of 6 RnR-based applications from the recognition, mining, search and video processing application domains in 45nm technology. Our experiments demonstrate 1.06X-2.18X reduction in energy consumption with virtually no loss in output quality ($<$0.5%) at the application-level. The energy benefits further improve up to 2.38X and 2.5X when the quality constraints are relaxed to 2.5% and 5% respectively.

I. Introduction

Applications from several domains such as recognition, data mining, analytics, vision, search, graphics, multimedia, and signal processing exhibit the property of *intrinsic application resilience*, which is the ability to produce outputs of acceptable quality even when the underlying computations are executed in an imprecise or approximate manner. This intrinsic resilience arises from several factors: (i) the robustness of these applications to noisy/redundant input data, (ii) the lack of a unique, golden answer, or the inability of humans to perceive minor variations in the application output, and (iii) the statistical and iterative nature of their computations [1], [2]. Approximate computing is an emerging design paradigm that leverages the intrinsic resilience of applications to execute computations approximately, and more efficently, leading to improvements in energy or performance [2]–[15].

In this work, we explore approximate computing in the context of a key computational pattern called *Reduce-and-Rank* (*RnR*), which is prevalent in a variety of existing and emerging application domains such as multimedia processing, recognition, data mining, computer vision, and search, among others. Given an input vector and a set of reference vectors, the *RnR* kernel performs a vector reduction operation (*e.g.*, multiply-and-accumulate, Euclidean distance, L1 norm) between the input vector and each of the reference vectors to produce a set of reduction outputs. These reduction outputs are subsequently ranked to select a top subset of reference vectors for the current input. For example, in the well-known k-means clustering algorithm, distance computation between a point and all cluster centroids, followed by identification of the closest cluster, is an *RnR* computation. *RnR* computations frequently dominate the computational requirements of the applications in which they are used (50-92% of runtime in our benchmarks was in *RnR* kernels), and are hence attractive targets for optimization.

We develop two complementary strategies to approximate *RnR* kernels *viz., interleaved reduction-and-ranking* and *input similarity based approximation*. The first strategy, interleaved reduction-and-ranking, decomposes the reduction computation into multiple partial reductions and utilizes the intermediate partial reduction output to modulate the degree of approximation introduced in subsequent computations. Recall that the reduction output is used to rank and select a subset of reference vectors. Therefore, the partial reduction output can be used to predict whether the current reference vector would be likely to eventually appear in the selected subset. The second strategy, input similarity based approximation, leverages the spatial or temporal correlation of successive inputs (*e.g.,* pixels in an image or frames of a video) present in many applications. We utilize the similarity between the current and previous input to infer the degree to which the computations on the current input can be approximated. Note that these approximation strategies only identify which future computations could be approximated based on intermediate results, and hence can be used in conjunction with any approximate computing mechanism to improve energy efficiency. Towards this end, we utilize two popular approximate design techniques *viz.,* computation skipping [2], [3] and precision scaling.

A key feature of the proposed approximation strategies is that they naturally facilitate the design of quality-configurable systems. Quality configurability, or the ability to modulate the energy *vs.* accuracy trade-off at runtime, is highly desirable in approximate computing, as the extent to which computations can be approximated varies from application to application, and even across inputs for a single application [5]. To enable quality configurability in *RnR* kernels, we first define a kernel-level quality metric for *RnR* that correlates well with the application-level output quality and identify parameters within the proposed approximation strategies that modulate how aggressively future computations are approximated. We then develop a systematic run-time framework to modulate the parameters based on a

This work was supported in part by the National Science Foundation under grants CNS-0953468, CNS-1423290, and CNS-1018621.

specified *RnR* quality constraint, thereby achieving quality-configurable execution.

The key contributions of our work can be summarized as follows:

- We explore approximate computing in the context of a key computation pattern – *Reduce-and-Rank*– and propose two broad strategies, *interleaved reduction-and-ranking* and *input similarity based approximation*, which can be used with any approximation mechanism.
- To enable quality configurability, we identify parameters within these strategies that dynamically trade-off quality for efficiency and develop a run-time framework to modulate the parameters based on a given *RnR* quality constraint.
- We construct quality-configurable implementations for a suite of benchmarks and demonstrate that the proposed techniques result in significant improvement in energy (1.06X-2.18X) for negligible ($<$0.5%) quality loss.

The rest of the paper is organized as follows. Section II describes related work in the field of approximate computing. Section III motivates the need for quality configurability in approximate computing. Section IV describes in detail the design approach and methodology for constructing quality configurable *RnR* kernels. Sections V and VI describe the experimental methodology and the results, respectively. Section VII summarizes and concludes the paper.

II. Related Work

Approximate computing has been explored at various layers of design abstraction including software [2]–[4], architecture [6], [7], and circuits [8]–[16]. Here, we restrict our discussions to circuit level approximation techniques, since that is the focus of our work.

Approximation techniques at the circuit level can be classified into two broad categories: i) *static*, and ii) *dynamic* approximation techniques. Static approximation techniques fix the nature and degree of approximation at design time. As a result, they result in widely varying output qualities for different inputs. Many static approaches focus on specific arithmetic units such as adders [11] and multipliers [14]. They have been generalized into approximate logic synthesis techniques based on modification of min-terms in two-level synthesis [17], circuit simplification by injecting stuck-at-faults at internal nodes [8], and exploiting the use of don't cares to simplify circuits using traditional boolean optimizations [9]. A probabilistic pruning technique for low activity circuit paths has been proposed to construct inexact circuits [10]. [12] demonstrates the use of dataflow graphs to identify and approximate adders which are relatively less significant. Recently, ABACUS [15] proposed a systematic framework for designing approximate circuits at the behavioral level. It first forms an abstract syntax tree (AST) from the input RTL, and subsequently approximates the AST using a variety of high-level transformations.

Dynamic approximation techniques have the ability to tune the degree of approximation based on the input. One such early approach is adaptive voltage over-scaling [16]. [4], [5], [7] vary the degree of approximation at run-time, however they are able to do so only at a very coarse granularity. This is due to the fact that quality modulation, when done frequently and at finer granularity, incurs a huge overhead which undermines most of the energy savings achieved from approximation. [18] is a low overhead mechanism of modulating the quality of approximate circuits at a finer granularity, however this technique is limited to an MPEG encoder only.

Our work differs from previous work along two key dimensions. First, we focus on a specific, but broadly used, computation pattern (*RnR*), and identify approximation strategies that leverage its characteristics. Second, the proposed approximation strategies are inherently dynamic in that they utilize intermediate results to infer opportunities for approximating future computations. Thus, they enable quality configurability at minimal overheads. Leveraging this attribute, we propose a run-time framework for quality-configurable execution of *RnR* computations.

III. A Case for Quality Configurable Systems

The primary objective of our work is to design quality configurable systems that are equipped with the ability to modulate the extent to which computations are approximated at run-time. The need for quality configurability stems from two key factors: (i) the intrinsic resilience of an application varies based on the context in which its outputs are consumed, and (ii) even for a given application output quality, there is significant heterogeneity in the degree to which different inputs to the application can be approximated. In this section, we demonstrate the need for quality configurability using an example. We consider a popular classification algorithm, k-Nearest Neighbors (k-NN), and use it in two different application contexts - digit recognition and eye detection.

Given an input, the k-NN algorithm identifies 'k' vectors from the training set that are closest to the input (*i.e., top-k*). It then assigns the input to the class that is most common amongst the neighbors. A typical implementation of k-NN would maintain a list of *top-k* vectors and iteratively insert vectors into the list based on its proximity to the input. If the distance of the vector is greater than those in the *top-k* list, it is simply discarded. Since the discarded vectors do not affect the application output, the number of updates to the *top-k* vector list provides a measure of the intrinsic resilience of the algorithm.

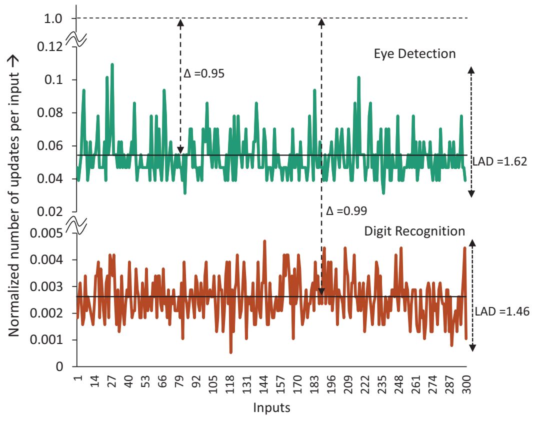


Fig. 1: Variation in the fraction of training vectors that update the top-k list for digit recognition and eye detection applications

Figure 1 shows the fraction of training vectors that update the *top-k* list for each input vector in the test set. The important inferences from Figure 1 are:

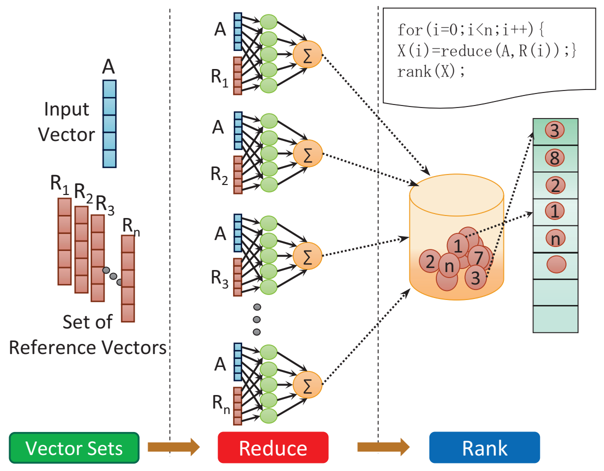
- The fraction of training vectors that update the *top-k* list is very small (5% for eye detection and $<$0.5% for digit recognition). This implies that the distance computation for a significant fraction ($>$95%) of the training vectors could be done approximately with no impact on application output. This underscores the significant potential for approximate computing.
- The degree of resilience varies across application contexts. For example, the number of updates in the case of digit recognition is $\sim$10X smaller compared to eye detection. Hence, the k-NN algorithm can be more aggressively approximated if it is used for digit recognition *vs.* eye detection.

- There is a significant variation (largest absolute deviations of 1.46X for digit recognition and 1.62X for eye detection) in the number of updates across inputs within each application. Therefore, inputs with a lower fraction of updates are more amenable to approximations.

Hence, quality configurability is highly desirable in approximate computing systems. However, the key challenge is to identify the extent to which computations can be approximated at runtime. In this work, we address this challenge based on the insight that intermediate results (*e.g.*, partial distance in the case of k-NN) can be utilized to guide the degree of approximation for the remaining computations.

IV. Quality Configurable Reduce-and-Rank: Design Approach and Methodology

In this work, we consider the design of quality configurable systems in the context of a commonly used computation pattern, namely *Reduce-and-Rank* (*RnR*). *RnR* computational kernels are prevalent in a number of emerging and existing application domains such as recognition, data-mining, vision, search, and video processing. Some prominent examples of *RnR* algorithms include k-Nearest Neighbors, K-Means clustering, MPEG encoder, Generalized Learning Vector Quantization, Image Segmentation, and the Sobel operator. In these applications, *RnR* constitutes a significant fraction of the runtime, as shown in Table I. The computation involved in an *RnR* kernel is illustrated in Figure 2. It takes an input vector (A) and a set of reference vectors ($R_1 \ldots R_N$) as its input and produces a ranked list of the reference vectors as its output. It consists of two steps. The *reduce* step performs vector reduction of the input vector with each reference vector to produce a list of scalars. These scalars are then fully or partially (top-k scalars) sorted in the *rank* step to produce the output. The k-NN algorithm, described in Section III, is an example of an algorithm comprising the *RnR* kernel. In this case, the input vector is *reduced* with each training vector to compute a list of distances, which are then *ranked* based on the distance values to obtain the *top-k* nearest training vectors. In this section, we describe the design approach and the methodology used to construct quality configurable *RnR* kernels.



```
for(i=0;i<n;i++){
X(i)=reduce(A,R(i));}
rank(X);
```

Fig. 2: *Reduce-and-Rank* kernel operation

A. Quality metric for RnR kernel

Quality configurability requires the specification of discrete quality bounds *a priori*, to be used for deriving the approximate versions of the applications. For *RnR* based applications, it is more convenient for the designer to specify the qualities in terms of the ranked outputs of the common *RnR* kernel rather than to deal with various application level qualities. The *RnR* quality can eventually be correlated empirically to the application quality. This leads us to propose a quality metric for the standalone *RnR* kernel. Most *RnR* based applications are concerned with identifying just a top fraction of the ranked elements, or the *top-k* elements. The relative ordering among these *top-k* elements is insignificant from the point of the application's correctness. *RnR* applications that involve finding only the minimum or maximum value of a metric can be thought of as a special case where $k = 1$.

Therefore, we define Q_{RnR}, the *Reduce-and-Rank* kernel quality metric as:

$$Q_{RnR} = \frac{\sum_{i=1}^{k}(rank_{app}(i) - k)}{k}, \forall rank_{app}(i) > k \qquad (1)$$

$rank_{app}(i)$ is the approximate rank of the i^{th} ranked entity in the accurate version of the algorithm. Q_{RnR} records the average deviation in rank of those elements that were in the *top-k* list in the accurate version, but fall outside the *top-k* list in the approximate version. We later show that this metric can be easily implemented in hardware with minimum overhead.

B. Quality Configurable RnR kernel: Design Approach

In this subsection, we explain two broad strategies that can be used to design quality configurable *RnR* kernels.

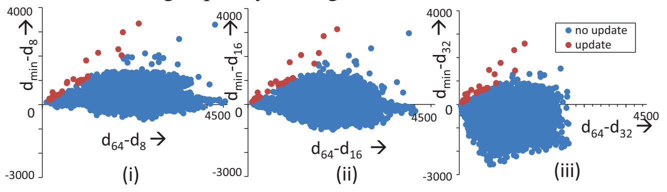


Fig. 3: Partial reduction states

1) Interleaved Reduction-and-Ranking: The first approximation strategy utilizes intermediate results computed in the application to glean insights that can be used to guide the approximation process for subsequent computations. Quality knobs controlling the degree of approximation can also utilize these internal results to dynamically configure the quality mode when required. For high dimensional datasets, the *RnR* kernel spends a major portion of its execution time in performing redundant computations that do not contribute to the correct final output. An effective technique to remove this redundancy is to split or decompose the reduction operation into multiple sequential stages which can be interleaved with the ranking step to deduce the nature and extent of approximations for (or completely eliminating) the remaining reduction operations. We again use the example of k-NN to explain this concept. In the case of k-NN, partial reduction states can be derived by computing the distance between the input and the reference vectors using only a small fraction of the total vector dimensions. Once the magnitude of the partial reduction exceeds the current k^{th} minimum distance, that reference vector can never update the *top-k* list, since the growth of the distance computation can only monotonically increase with dimensions. Hence, any remaining reduction operations for that reference vector can be safely terminated without any loss in output quality. Fig. 3 plots these intermediate reduction states for 10000 random distance computations used in digit recognition. The total number of dimensions of the vectors is 64 and the partial reduction states are computed after 8,16, and 32 dimensions. All the points that are below the x-axis have already exceeded the threshold to update the *top-k* list, and can be safely rejected. Note that the monotonic growth of the distance value causes more points to be rejected at higher dimensions. Fig. 3 also shows that even among those vectors for which the reduction operations can not be terminated early, only a very small fraction actually update the *top-k* list. The distance computations for remaining reference vectors (blue points above the x-axis) can be approximated with minimal effect on quality. Without any loss of generality, we apply precision scaling [6] to approximate these reduction operations. Precision scaling is an approximation technique which ignores a certain number of the least significant bits of the input operands during computation.

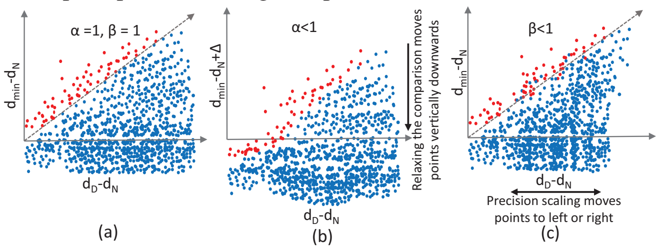


Fig. 4: Effect of α, β on the points

We define two parameters, namely *relaxation factor* (α) and *precision control* (β) that control the manner in which the partial reduction values are utilized to dynamically tune the degree of computation skipping and precision scaling for different inputs to the *RnR* kernel. α relaxes the strict comparison of the reduction value to the *top-k* elements, causing a larger number of points to be skipped as shown in Fig. 4(b). Precision control (β) controls the extent to which the points move horizontally as a result of precision scaling shown in Fig. 4(c). Combination of these two approximation techniques can cause points to either miss or falsely update the *top-k* list resulting in quality degradation. However, the percentage of points that update the *top-k* list being very small allows these two techniques to produce highly energy-efficient hardware implementations with minimal accuracy loss.

2) Input Similarity based Approximations: In image and video processing applications, we can leverage the spatial correlation or similarity between adjacent input points (or temporal locality between frames of a video) to approximate or entirely skip processing a portion of the inputs. This approximation technique is generic enough to be applicable to any application where the inputs are correlated and arrive in a specific order. To explain how this technique works, we take an example of the image segmentation application which uses the *RnR* based K-Means clustering algorithm to partition an image into K classes. This algorithm allocates a pixel to a particular cluster (or class) if the distance between the pixel vector and the cluster's center is the least among all the K clusters. Fig. 5 presents the correlation plot for a test image, which shows that a large number of adjacent pixels frequently belong to the same class. We argue that if the distance from the designated cluster's center is within a small fraction γ (*correlation factor*) of the cluster radius, then due to spatial locality, the adjacent s pixels also belong to the same cluster, and hence the reduction operations for the correlated adjacent points can be skipped. Discarding similar inputs can result in considerable reduction in application energy with minimal effect on output quality. Here, internal variables such as distances from the cluster centers and cluster radii control the extent to which we can skip processing the adjacent pixels. The parameter γ controls the way in which these variables are employed to modulate the quality knob (number of inputs to approximate or skip) for quality configurable operation across inputs.

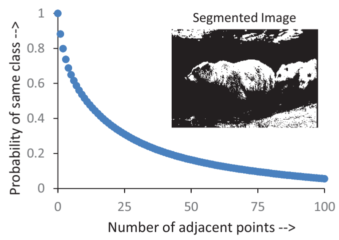


Fig. 5: Correlation graph

3) Run-time calibration of (α, β), and γ: The proposed approximation strategies enable *Reduce-and-Rank* kernels to operate at various quality modes by tuning the internal quality knobs for computation skipping and precision scaling. However, in order to select the right quality configuration, the application first needs to identify the scope of approximations for the present input dataset. We constructed an automatic framework (along the lines of [4], [7]) that first learns the output qualities for different configurations of the parameters over a range of inputs, and subsequently uses this knowledge to set the quality knobs during actual operation. The first phase is termed as the *calibration phase*, where the system automatically derives the estimates of *RnR* quality and energy for different values of the parameters. The second stage is known as the *evaluation phase*, in which the application uses the appropriate (α, β) (or γ) settings and the imposed quality constraint to dynamically regulate the quality knobs according to the approximation potential of each input. The calibration period is invoked periodically so that the quality knobs can efficiently track any variations in the scope of approximations for widely varying inputs.

C. Methodology

This section details how we automate the modulation of the quality knobs on the basis of internal states and output quality bounds. As mentioned above, the overall operation of the proposed system consists of two phases: (i) *Calibration Phase*, and (ii) *Evaluation Phase*.

Algorithm 1 Extraction of parameters at *calibration phase*

```
Input: $Q_{RnR}$ Bound: $Q$, List of quality pairs: $[\alpha, \beta]_{list}$
Output: $(\alpha, \beta)$ for quality bound $Q$: $(\alpha_Q, \beta_Q)$
Begin
    Initialize: $i = j = 0$, $Q_{curr} = 1$
    for each $(\alpha, \beta) \in [\alpha, \beta]_{list}$ do
        $Q_{bin}(\alpha, \beta) = count\_updates(\alpha, \beta)$
        $E_{cnt}(\alpha, \beta) = count\_points(\alpha, \beta)$
    end for
    while $(Q_{curr} \geq Q)$ do
        $(\alpha_Q, \beta_Q) = (\alpha_r[i], \beta_r[j])$ # $\alpha_r, \beta_r$ sorted by decreasing quality
        $Q_\alpha = get\_rnr\_qual(Q_{bin}(\alpha_r[i+1], \beta_r[j]))$ # normalized Q
        $E_\alpha = get\_rnr\_en(E_{cnt}(\alpha_r[i+1], \beta_r[j]))$ # normalized E
        $Q_\beta = get\_rnr\_qual(Q_{bin}(\alpha_r[i], \beta_r[j+1]))$
        $E_\beta = get\_rnr\_en(E_{cnt}(\alpha_r[i], \beta_r[j+1]))$
        $R_\alpha = \frac{E_{orig} - E_\alpha}{Q_{orig} - Q_\alpha}$, $R_\beta = \frac{E_{orig} - E_\beta}{Q_{orig} - Q_\beta}$
        if $(R_\alpha > R_\beta)$ then
          if $(Q_\alpha \geq Q)$ then $Q_{curr} = Q_\alpha$; $i = i + 1$
          else if $(Q_\beta \geq Q)$ then $Q_{curr} = Q_\beta$; $j = j + 1$ else break
        else
          if $(Q_\beta \geq Q)$ then $Q_{curr} = Q_\beta$; $j = j + 1$
          else if $(Q_\alpha \geq Q)$ then $Q_{curr} = Q_\alpha$; $i = i + 1$ else break
        end if
    end while
    return $(\alpha_Q, \beta_Q)$
End
```

1) Calibration Phase: During the calibration phase, the application figures out the optimal values for the parameters to be used during the actual *RnR* operation. Algorithm 1 gives a conceptual overview of how the system can automatically evaluate the appropriate (α, β) for a specified Q_{RnR}. The same method can also be extended to obtain γ as well. Here, the key concept is that once the controller knows the *RnR* quality and energy estimates (E_{RnR}) for all (α, β) pairs, it can prune the search space efficiently and obtain the best (α, β) for a Q_{RnR} bound through gradient descent. Q_{RnR} and E_{RnR} estimates are acquired with the help of counters embedded in the *RnR* hardware known as *quality bins* (Q_{bin}) and *energy counters*

(E_{cnt}), respectively. It is quite straightforward to estimate E_{RnR} from E_{cnt}, which registers the total number of points either skipped or precision scaled. Q_{bin} records the total number of times the *top-k* list (or, $RankList$) gets updated. It can be shown that Q_{RnR} can be estimated from the knowledge of the missed updates. If x is the number of missed updates, then the maximum Q_{RnR} can be either $\sum_{i=1}^{x} i/k$ if $x < k$, or $\sum_{i=x-k+1}^{x} i/k$ otherwise. This x can be derived by subtracting the Q_{bin} counts from the original number of updates. Since the calibration phase is invoked periodically, the controller uses Q_{bin} and E_{cnt} to maintain cumulative counts across calibration phases, thus learning from both past and present training inputs.

Algorithm 2 *Reduce-and-Rank* operation at *evaluation phase*

```
Input: Set of reference vectors: $P$, Set of input vectors: $N$, Correlation = $corr$
Output: List of top-k elements for each $\vec{n} \in N$: $RankList_n$
Begin
  Initialize: $MinList(1...k) = MAXVAL$
  for each $\vec{n} \in N$ do
    $skip\_point = 0$
    # check correlation among inputs #
    if $(corr == 1)$ then
      $G_{n-1} = get\_class(RankList_{n-1})$
      if $((comp\_class\_thresh(MinList_{n-1}(k), \gamma, T_{G_{n-1}}) == 1)$ then
        $[G_n : G_{n+s-1}] = G_{n-1}$; $n = n + s - 1$ #skip $s$ inputs
        $skip\_point = 1$
      end if
    end if
    if $(skip\_point == 0)$ then
      for each $\vec{p} \in P$ do
        if $(D > D_{th})$ then
          # decomposed reduction #
          $M^n_{C_1}(p) = Red_0(n, p, [0 : C_1])$ #partial redn. till $C_1$ dims
          for each $c \in C$ do
            if $(M^n_c(p) \geq \alpha_c MinList_n(k))$ then
              $M^n_D(p) = MAXVAL$; break #skip dimensions c to D
            else
              for each $\beta_{psc} \in \beta_{set}$ do
                if $(M^n_c(p) \leq \alpha_c \beta_{psc} MinList_n(k))$ then
                  $M^n_{c+1}(p) = M^n_c(p) + Red_{X_{\beta_{psc}}}(n, p, [C_c : C_{c+1}])$; break
                end if
              end for
            end if
          end for
        else $M^n_D(p) = Red_0(n, p, [0 : D])$
        end if
        $Rank\_topk(p, M^n_D(p), MinList_n)$
      end for
      return $RankList_n$
    end if
  end for
End
```

2) Evaluation Phase: This is the operational phase for *Reduce-and-Rank* applications, where the system tunes the quality knobs controlling the degrees of approximation for different inputs to maximize the energy benefits for a specified quality constraint. Algorithm 2 presents a unified pseudo-code integrating both interleaved reduction-and-ranking and input similarity based approximation strategies. The algorithm first checks whether the present input exhibits sufficient correlation with immediately preceeding inputs or not. In case it does, it decides to skip processing it. Note that the designer needs to pre-specify whether the present dataset can have correlation or not and accordingly sets the $corr$ input. If the present vector is not skipped, the algorithm decides whether to $decompose$ the reduction operation on the basis of the total number of dimensions D. The reduction function is denoted by Red_{PSc}, where the subscript denotes the degree of precision scaling. C denotes the set of dimensions where the partial reduction states are computed. Once the final reduction values are evaluated, the ranking function $Rank\_topk$ is invoked, which lists the *top-k* elements in ascending order (or, descending order as required). Although different values of α_c can be selected for different partial reduction states, they are assumed to be constant fractions of a single α. Similarly, all values of β_{psc} are constant multiples of β. This ensures that the search space for finding the appropriate parameters is small. The degrees of precision scaling are represented by $X_{\beta_{psc}}$, where $X_{\beta_c} > X_{\beta_d}$ *iff* $\beta_c > \beta_d$, since a higher partial reduction value denotes a lower probability to update the $RankList$.

V. Experimental Methodology

We evaluate the effectiveness of the proposed design techniques on a wide variety of benchmarks as listed in Table I. A combination of interleaved reduction-and-ranking and input similarity based approximation can be applied for applications such as MPEG encoding, where the inputs are both high-dimensional and spatially correlated. The values of α, β, γ, and s were determined empirically for each of the applications. The applications were implemented in hardware with additional registers, counters, and control logic for automatically tuning the quality knobs during the calibration phase. These were then synthesized using Synopsys Design Compiler and mapped to 45nm based Nangate Open Cell library. Finally, Synopsys Power Compiler [19] was used to estimate the power and energy consumption of the applications.

VI. Experimental Results

This section presents the results of various experiments conducted using the 6 benchmarks shown in Table I. The results prove the efficacy of our proposed framework by demonstrating a significant reduction in application energy consumption while satisfying the given quality constraint.

A. Application-level energy benefits

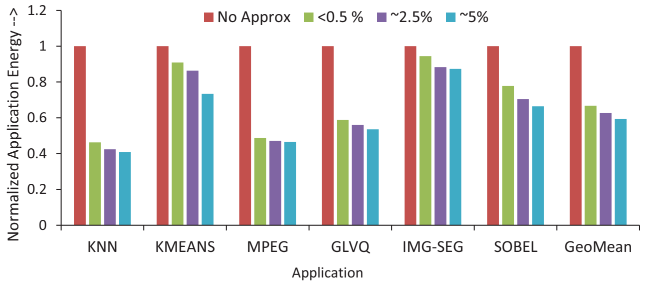


Fig. 6: Energy consumption for different application-level quality constraints

Fig. 6 plots the application-level energy consumption of the evaluation benchmarks for different quality specifications. The energy values are normalized with respect to the energy consumption of the original accurate applications. The results denote the extent to which *RnR* kernel approximations are translated into application level energy benefits. As shown in the figure, our techniques results in a reduction of 1.06X - 2.18X in application energy consumption for virtually no quality degradation (<0.5%). In addition the energy reduction jumps to a range of 1.14X - 2.38X and 1.15X - 2.5X for 2.5% and 5% quality loss, respectively. The additional hardware and the calibration phase impose an energy overhead of 10% for the quality configurable *RnR* applications compared to the accurate versions of the applications, which are accounted for in the results shown in Fig. 6.

TABLE I: Benchmarks used for evaluation

Application	**Algorithm**	***RnR* kernel**	**%Runtime in *RnR* Operation**	**Dataset (Vector Dims.)**	**Method Used**	**Quality Metric**
Optical Character Recognition	k-Nearest Neighbors (KNN)	Computes distance & selects the *top-k* closest reference vectors	92	OCR Digits (64)	Interleaved RnR	% Classification Accuracy
Eye Detection	Generalized Learning Vector Quantization (GLVQ)	Computes distance & finds the nearest & farthest reference vectors	89	Image set from NEC labs (512)	Interleaved RnR	% Classification Accuracy
Video Streaming	MPEG encoder (MPEG)	Derives minimum sum of absolute differences (SAD) between 2 macroblocks	85	Sample videos from Xiph.org (256)	Interleaved RnR, Input Similarity	% PSNR Degradation
Optical Character Clustering	K-Means Clustering (KMEANS)	Computes distance from cluster centroids & selects the nearest cluster	66	OCR Digits (64)	Interleaved RnR	Mean distance from centroids
Edge Detection	Sobel Operator (SOBEL)	Computes convolution & does gradient approximation with min and max values	50	Standard Test Images (9)	Input Similarity	Mean SAD wrt. accurate output
Image Segmentation	K-Means Clustering (IMG-SEG)	Computes distance from cluster centroids & selects the nearest cluster	66	Berkeley Dataset (3)	Input Similarity	Mean distance from centroids

B. System behavior during runtime

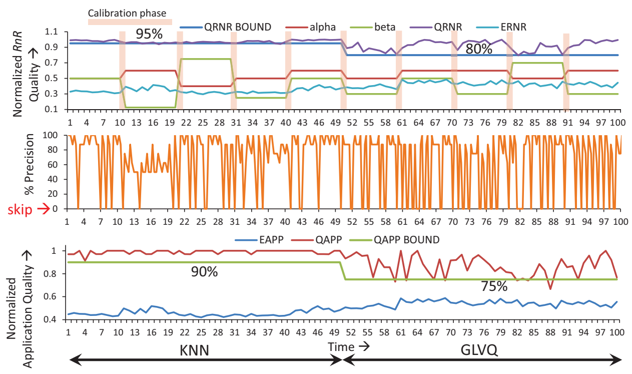


Fig. 7: Quality/Energy vs. Time

Fig. 7 shows the dynamic modulation of precision scaling for each of the inputs, thereby resulting in the variation of *RnR* and application energy. Additionally, it also shows how the controller automatically adapts α, β to maintain quality bounds across a range of inputs over time. All the quality and energy metrics are normalized with respect to the quality and energy metrics of the original accurate applications. In this case, we run two applications k-NN and GLVQ back to back with different Q_{RnR} bounds at 95% and 80%, which can be empirically mapped to application-level quality bounds of 90% and 75%, respectively . Note that a precision value of *0* indicates that future computations for that vector are completely skipped. The graph shows that our design technique successfully satisfies the output quality bound across different inputs while yielding substantial energy benefits.

C. Design space exploration

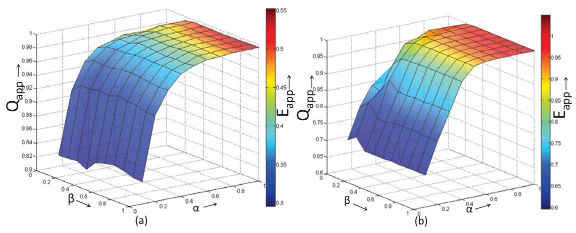


Fig. 8: Application Quality *vs.* Application Energy

Finally, Fig. 8 presents the variation of application quality (Q_{app}) and application energy (E_{app}) with α and β for two applications, namely (a) KNN and (b) KMEANS. Q_{app} and E_{app} are again normalized with respect to the values for the accurate versions of the applications. This *Q-E* trade-off curve is automatically extracted during the calibration phase to correctly tune the parameters (α, β) (or γ). As expected, both Q_{app} and E_{app} gradually decrease with decreasing α and β, since a larger number of vectors are either skipped or precision scaled. Fig. 8 also clarifies the concept of gradient descent in Algo. 1. Since there can be multiple possible (α, β) configurations for a particular Q_{app} (or an equivalent Q_{RnR}) bound, Algo. 1 always selects the (α, β) pair that consumes the lowest energy.

VII. Conclusion

In this paper, we presented a novel design technique to construct quality configurable *Reduce-and-Rank* kernel based applications using insights derived from intermediate results within the applications to drive the approximation strategy. In addition, we also proposed an automatic run-time framework that enables the application to dynamically self-tune the quality knobs according to the current scope of approximation. When evaluated over a set of benchmarks, this technique results in significant energy savings while also adhering to the given quality constraint across different inputs.

References

[1] M. A. Breuer. Multi-media applications and imprecise computation. In *Proc. Euromicro Conf. on Digital System Design*, pages 2–7, Sept. 2005.
[2] S. T. Chakradhar et al. Best-effort computing: Re-thinking parallel software and hardware. In *Proc. DAC*, pages 865 –870, June 2010.
[3] S. Sidiroglou-Douskos et al. Managing performance vs. accuracy trade-offs with loop perforation. In *Proc. ESEC/FSE*, pages 124–134, 2011.
[4] W. Baek et al. Green: A framework for supporting energy-conscious programming using controlled approximation. In *Proc. PLDI*, 2010.
[5] V. Chippa et.al. Dynamic effort scaling: Managing the quality-efficiency tradeoff. In *Proc. DAC*, pages 603–608, 2011.
[6] S. Venkataramani et al. Quality programmable vector processors for approximate computing. In *Proc. MICRO-46*, pages 1–12, 2013.
[7] M. Samadi et al. Sage: Self-tuning approximation for graphics engines. In *Proc. MICRO-46*, pages 13–24, 2013.
[8] D. Shin et al. A new circuit simplification method for error tolerant applications. In *Proc. DATE*, pages 1 –6, Mar. 2011.
[9] S. Venkataramani et al. Salsa: systematic logic synthesis of approximate circuits. In *Proc. DAC*, pages 796–801, 2012.
[10] A. Lingamneni K. Palem et al. Energy parsimonious circuit design through probabilistic pruning. In *Proc. DATE*, pages 1 –6, Mar. 2011.
[11] A. B. Kahng et al. Accuracy-configurable adder for approximate arithmetic designs. In *Proc. DAC*, pages 820–825, 2012.
[12] Z. Kedem et al. Optimizing energy to minimize errors in dataflow graphs using approximate adders. In *Proc. CASES*, pages 177–186. ACM, 2010.
[13] A. Ranjan et al. Aslan: Synthesis of approximate sequential circuits. In *Proc. DATE*, pages 364:1–6, 2014.
[14] P. Kulkarni et al. Trading accuracy for power with an underdesigned multiplier architecture. In *Proc. VLSI Design*, pages 346 –351, Jan. 2011.
[15] K. Nepal et al. Abacus: A technique for automated behavioral synthesis of approximate computing circuits. In *Proc. DATE*, pages 361:1–6, 2014.
[16] P. K. Krause et al. Adaptive voltage over-scaling for resilient applications. In *Proc. DATE*, pages 1 –6, March 2011.
[17] D. Shin et al. Approximate logic synthesis for error tolerant applications. In *Proc. DATE*, pages 957–960, Mar. 2010.
[18] A. Raha et al. A power efficient video encoder using reconfigurable approximate arithmetic units. In *Proc. VLSI Design*, pages 324–329, 2014.
[19] Design Compiler, Synopsys Inc.