# ApproxMA: Approximate Memory Access for Dynamic Precision Scaling

Ye Tian, Qian Zhang, Ting Wang, Feng Yuan, and Qiang Xu
CUhk REliable computing laboratory (CURE)
Department of Computer Science, The Chinese University of Hong Kong
Shatin, N.T., Hong Kong
tianye,qzhang,twang,fyuan,qxu@cse.cuhk.edu.hk

## ABSTRACT

*Motivated by the inherent error-resilience of emerging recognition, mining, and synthesis (RMS) applications, approximate computing techniques such as precision scaling has been advocated for achieving energy-efficiency gains at the cost of small accuracy loss. Most existing solutions, however, focus on the approximation of on-chip computations without considering that of off-chip data accesses, whose energy consumption may contribute to a significant portion of the total energy. In this work, we propose a novel approximate memory access technique for dynamic precision scaling, namely* ApproxMA*. To be specific, by taking both runtime data precision constraints and error-resilient capabilities of the application into consideration, ApproxMA determines the precision of data accesses and loads scaled data from off-chip memory for computation. Experimental results with mixture model-based clustering algorithms demonstrate the efficacy of the proposed methodology.*

## Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems—*Design studies*

## Keywords

Approximate Computing, Precision Scaling, Memory Access

## 1. INTRODUCTION

By trading off computation quality with computational effort, approximate computing [1–5] is a promising energy-efficient technique for emerging Recognition, Mining and Synthesis (RMS) applications due to their inherent error-resilience characteristics. Firstly, they are usually used to process large amounts of data that are often noisy and redundant; Secondly, there is usually no specific "golden" output value but rather many "acceptable" outputs; Finally, the algorithms used in many of these applications are stochastic in nature and often resort to error-resilient methods for solution-finding.

Precision scaling, which decreases the operand bit-width in computations according to application quality requirements, is one of the most effective approximate computing techniques [6–8]. Considering the fact that the computation quality requirement may vary significantly at runtime, various dynamic precision scaling [9, 10] techniques have been presented in the literature, which adaptively adjust operand bit-width to improve energy efficiency under quality constraints.

For RMS applications that are used to process a large volume of data, there are inevitably frequent interactions between off-chip memory and on-chip computational units, and the energy needed for such communication can be much higher than that of computations. Intuitively, if we could selectively load certain most significant bits (instead of all the bits) of data from off-chip under quality constraints, significant energy-efficiency gains can be achieved. Most existing precision scaling techniques, however, only target at energy-efficient on-chip computations without much consideration of off-chip memory accesses.

Motivated by the above, in this work, we propose an approximate memory access framework, namely *ApproxMA*, for dynamic precision scaling with emphasis on off-chip memory accesses, and we validate its effectiveness on mixture model-based clustering problem. Mixture model-based clustering [11] assumes that data were generated by a mixture of models and tries to recover the original model from the data, which provides great flexibility for fitting any data set according to a particular distribution. It has a wide range of applications [12]. For instance, it is the essential part in grouping products and customers in massive retail datasets, gene sequence analysis to find genes that work together, and image segmentation and denoising in image/video processing. Mixture model-based clustering problem is inherently error-resilient and needs large amount of data accesses from off-chip memory. The computation demand for this task is usually quite high and hence how to improve its energy-efficiency is of great interest.

The remainder of this paper is organized as follows. Section 2 provides related works and motivates this paper. Section 3 demonstrates mechanism of approximate memory access for dynamic precision scaling. Case study of mixture model-based clustering with ApproxMA is then presented in Section 4. Finally, Section 5 concludes this paper.

## 2. RELATED WORKS AND MOTIVATION

Precision scaling is a commonly used approximate design technique, in which the bit-width (precision) of the input operands is modulated for energy efficiency. Many research efforts have been dedicated to precision scaling for tradeoff energy and quality in the literature [6, 8, 9, 13–15]. For example, QUORA [6] applies dynamic precision scaling into processing elements (PEs) with error monitoring and compensation to facilitate quality-programmable execution. A

*GLSVLSI'15,* May 20–22, 2015, Pittsburgh, PA, USA.
Copyright © 2015 ACM 978-1-4503-3474-7/15/05$15.00.
http://dx.doi.org/10.1145/2742060.2743759.

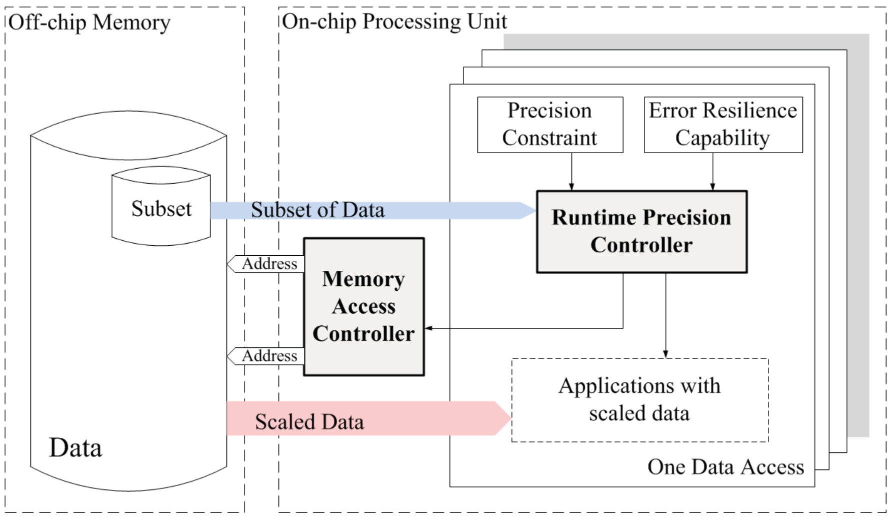

**Figure 1: The Framework of Approximate Memory Access.**

significant amount of energy is saved by adjusting the precision based on application quality constraints. Analysis of the intermediate variables has been proposed to set the bit-width of the input data under a given error bound of the computation [7]. In [8], word-length tunable architecture for OFDM (Orthogonal Frequency Division Multiplexing) Demodulator determines data word-length at runtime based on the observed error of the system output. [9, 13] optimize the word-length used in the operation according to the time-varying environment of wireless communication systems. There were also other works that apply precision scaling for FPGAs [14] and GPUs [15]. All the above works only apply precision scaling on the processing units.

In big-data era, RMS applications require to deal with extremely large size of dataset, thereby requiring frequent data communications between off-chip memory and on-chip computational units. On the other hand, due to much longer bus length and larger on-board parasitic capacitance, the energy consumption of loading data from off-chip memory is up to 19x comparing with that from on-chip memory [16]. Therefore, off-chip memory accesses could be the dominant factor of the total energy consumption for these communication-intensive RMS applications.

Motivated by the above, we propose an approximate memory access framework for dynamic precision scaling, namely *ApproxMA*. One related work is presented in [17], wherein the authors apply lossy compression for data transferred between the GPU and its off-chip memory, but it requires microarchitectural changes to the system.

## 3. THE PROPOSED FRAMEWORK

The proposed ApproxMA framework is presented in Fig. 1, which is comprised of *runtime precision controller* and *memory access controller*. For data accesses, runtime precision controller firstly generates the customized bit-width according to runtime quality requirements, and then memory access controller loads the scaled data from off-chip for computations. Although consequent computations also bring energy savings, our work mainly focuses on the communications with scaled data.

### 3.1 Runtime Precision Controller

Our runtime precision controller works based on the principle that it is not necessary to perform fully-accurate computation with approximate computing. On the one hand, lower precision data does not always lead to functional error. On the other hand, as many RMS applications are inherently error-resilient, certain amount of functional errors are acceptable, which can sometimes be recovered and have no influence on the final output quality. We denote this property as *error resilience capability* of the algorithm.

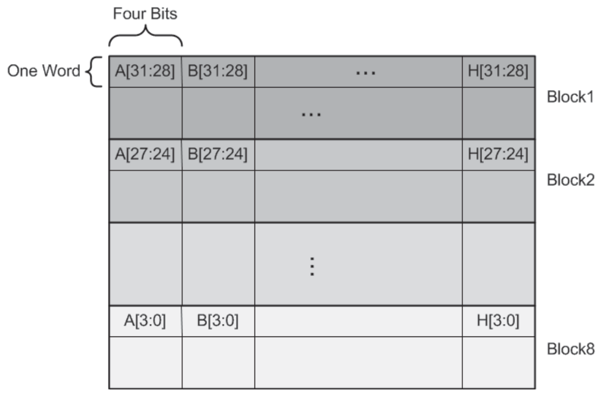

**Figure 2: Data Format in Off-chip Memory**

By analyzing subset of data and/or intermediate computational results, runtime precision controller calculates precision constraints (when there will be a functional error) and error resilience capability (how many functional errors are tolerable), and then decides the required bit-width for the current data access.

### 3.2 Memory Access Controller

To realize loading certain most significant bits of data from off-chip memory, we need to reorganize the data. Bits of the same significance in different words are combined to form new words and stored in off-chip memory. For example, assuming each word has 32 bits and we divide it into 8 parts, each of which then has 4 bits. We will store them in the format shown in Fig. 2, where $A, B, \ldots, H$ respectively represent the eight original words. The first block in off-chip memory stores the first 4 bits of all data, the second block stores the second 4 bits, and so on.

Knowing the starting address of dataset, the number of blocks and the bit-width of data to load, memory access

```
Input:
    startAdd (starting address of dataset),
    sizeOfData (original total size of dataset in off-chip memory),
    nOfBlock (number of blocks data are divided to),
    bitwidth (bit-width of data to load for current data access)
1 $offset = sizeOfData/nOfBlock$;
2 calculate $nOfBlockToLoad$ (number of blocks to load) with $bitwidth$;
3 while ($nOfBlockToLoad > 0$) do
4     Load data from the address $startAdd$ to the address $startAdd + offset - 1$;
5     $startAdd = startAdd + offset$;
6     $nOfBlockToLoad = nOfBlockToLoad - 1$;
7 end
```

**Algorithm 1:** Load Precision Scaled Data from Off-chip Memory

controller loads the corresponding scaled data from off-chip memory as shown in Algo. 1. For example, assuming we are able to read $m$ 32-bit original data words from off-chip memory once, if the current required precision is 16 bits, we can instead read $32 \times m \div 16 = 2m$ samples once. The access sequence is, bits [31:28] of the first $2m$ samples, then bits [27:24], [23:20] and [19:16] of these $2m$ samples. Having processed the first $2m$ samples, the processor then read the former 16 bits of the second $2m$ samples, and so on.

ApproxMA can be applied to general processor/accelerator architecture without necessarily changing the hardware, in which case, the access control of off-chip memory is conducted at the processor side with a software-based memory management unit MMU). Data reorganization needs to be performed before computation. Runtime precision controllers and memory access controllers can be realized by many kinds of algorithms in our framework depending on the specific applications and datasets. We will detail how ApproxMA can be applied for mixture model-based clustering problem in Sec. 4.

## 4. CASE STUDY FOR MIXTURE MODEL-BASED CLUSTERING PROBLEM

### 4.1 Preliminaries

Data clustering is commonly used to find the structure of a given dataset. To be specific, clustering algorithm groups alike samples based on certain distance metric (e.g., euclidean distance, probability-based distance) and finally outputs $k$ clusters to represent the whole samples. As no pre-knowledge on the different clusters or labels provided, clustering is a typical unsupervised learning procedure. Mixture model-based clusterings use certain models (e.g., distributions, centroids) for clusters and attempt to optimize the fitness between the samples and the models based on an objective function (e.g., maximize the likelihood or minimize the distances). In practice, each model can be mathematically represented by a parametric distribution, such as Gaussian (continuous) or Poisson (discrete). The entire dataset is therefore modeled by a *mixture* of these distributions.

Without loss of generality, let us consider Gaussian mixture model-based clustering (GMM). First, with given data $X = \{x_1, x_2, ...x_n\}$, we assume these samples are from $k$ Gaussians (i.e., specific *model*), and each of them can be uniquely identified by $G_i(\mu_i, \Sigma_i)$. Then, samples will be assigned with different labels based on the probability (i.e., specific *distance metric*) that they belong to each Gaussian. Next, all the parameters consisting such mixture models will be determined by optimizing the likelihood function (i.e., specific *objective function*) in an Expectation-Maximization (EM) manner. The algorithm is as follows:

```
$Repeat\ until\ convergence: \{$
    $E-Step(Calculate\ current\ measurement):$
        $For\ each\ data\ t\ and\ cluster\ j$
            $calculate\ the\ probability\ that\ t\ belongs\ to\ j$
    $M-Step(Optimize\ objective\ function):$
        $For\ each\ cluster\ j$
            $update\ \mu_j\ and\ \Sigma_j\ by\ Maximum\ Likelihood$
$\}$
```

In the E-step, it "guesses" the values of the probabilities (i.e., calculating the similarities for current model), and in the M-step, it updates the model parameters by Maximum Likelihood (i.e., optimizing objective function) based on the E-step's guesses. Although the distance metrics of various models are different, the above iterative EM-based learning procedure is the main-stream technique used for clustering.

### 4.2 Runtime Precision Controller

A functional error of clustering algorithm happens only if one sample is assigned to an incorrect cluster, and thus the computation with lower precision data does not always lead to functional errors if the relative distances hold. Precision constraints can be decided according to the lowest precision with no functional errors. From the point view of error resilience capability, an appropriate amount of functional errors (i.e., error rate[1]) are acceptable. The iterative nature of clustering algorithm is to continuously correct the mixture models and re-label all the samples. Therefore, the correctness of clustering results can be guaranteed in the later iteration with higher data precision. Such capability is oscillating until the convergence of clustering procedure.

#### 4.2.1 Overview of Runtime Precision Controller

In our proposed runtime precision controller, we represent the relationship between error rates and data precision levels with precision constraint table, and then precision is scaled according to the application's error resilience capability. In each iteration, both precision constraint table and error resilience capability are updated based on runtime information. However, it is a challenging problem, because (i) constraints and error resilience capabilities are all highly dependent on datasets and mixture models; (ii) modern datasets usually feature large amounts of high-dimensional data and there will be significant overhead to analyze all of them.

Fig. 3 depicts the overall flow of our runtime precision controller. It can be observed that *precision constraint table* is constructed from a subset of all the samples, and *error resilience capability* is obtained from intermediate computation results (i.e., classification membership changes between two iterations). The precision of data is determined based on the outputs of the above two blocks (i.e., constraint and relaxations). In addition, a *precision prediction calibration* module is applied to avoid the bit-width prediction error due

[1]The percentage of functional errors among all the samples

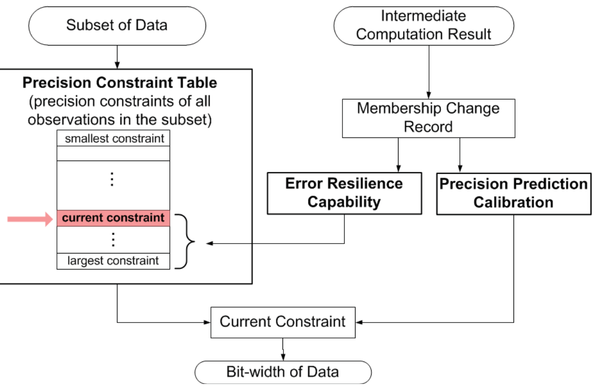
Subset of Data
Intermediate Computation Result
Precision Constraint Table
(precision constraints of all observations in the subset)
smallest constraint
current constraint
largest constraint
Membership Change Record
Error Resilience Capability
Precision Prediction Calibration
Current Constraint
Bit-width of Data

**Figure 3: Runtime Precision Controller**

to the incomplete or inexact information extraction (i.e., subset of samples). In the following parts, we will detail these functional blocks.

### 4.2.2 *Precision Constraint Table*

The precision constraint table lists the data precision constraints and the corresponding predicted error rates.

Suppose the subset we used to construct this table is denoted as

$$x = \{x_1, x_2, ...x_m\}.$$

For each sample, we would estimate the minimal precision requirement (i.e., bit-width bound) to avoid functional error, and then, construct the corresponding precision constraint table with precision requirements of all samples in the subset.

Let us take Fig. 4 as an example to illustrate how to estimate the minimal precision constraint for a single sample.

Given the number of clusters $k$, the clustering algorithm first computes the distances between the current sample and these $k$ clusters based on the specific distance metric used in the clustering algorithm, and then assigns it to the cluster with the smallest distance. That is, we get

$$d = \{d_1, d_2, ...d_k\},$$

where each $d_i$ indicates the distance between the current sample and the $i_{th}$ cluster, and if $d_s$ is the smallest value in $d$, the current sample will be assigned to $s_{th}$ cluster.

Based on the above, as long as the relative relationship between any other $d_i$ and $d_s$ is correct (i.e., $\forall d_i > d_s$, $i \neq s$), the functional error for this sample is avoided.

Specifically, for $d_j > d_s$ $(j \neq s)$, the tolerable relative error for keeping this relative relationship can be represented as

$$min(\frac{d_j - d_s}{d_j}, \frac{d_j - d_s}{d_s}) = \frac{d_j - d_s}{d_j} = 1 - \frac{d_s}{d_j}, j \neq s.$$

As there are totally $k-1$ relative relationships needed to be maintained, we get $k-1$ tolerable relative errors. Given $d_s$ must be smaller than any other $d_i$, the final relative error bound for labeling the current sample correctly must be the smallest one of these $k-1$ values.

For specific distance metric

$$d = f(x),$$

by conducting the above procedure, we get the final relative error bound on distance computation (i.e., bounds on $d$). Then, the bit-width constraint $c$ for the current data can be determined accordingly. That is, as long as we load $c$-bit of this sample, it will be classified correctly.

Then, for the entire selected subset, we get

$$C = \{c_1, c_2, ...c_m\},$$

wherein each $c_i$ indicates the bit-width constraint for the $i_{th}$ sample in the subset. After sorting these values in ascending order, we can predict the error rate with any $c_i$ as follows:

$$rate = \frac{No.(entries\ in\ C\ that\ is\ smaller\ than\ c_i)}{m}$$

Finally, we obtain the precision constraint table.

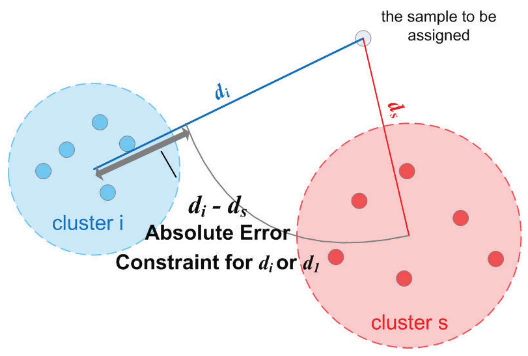
the sample to be assigned
$d_i$
$d_s$
cluster i
$d_i$ - $d_s$
Absolute Error Constraint for $d_i$ or $d_l$
cluster s

**Figure 4: Absolute Error Constraint for One Comparison**

### 4.2.3 *Error Resilience Capability*

Membership of a sample depicts which cluster the sample belongs to. We define the change of memberships in one iteration as the percentage of samples whose membership changes comparing with that of last iteration. The change of membership tends to decrease along with the algorithm approaching convergence and will finally reach zero. This value reflects the converging rate of the clustering algorithm, and thus we leverage it as the indicator of error resilience capability. For example, if 30% samples change their memberships, the error resilience capability is set as $\alpha \times 30\%$, which means we can tolerate at most $\alpha \times 30\%$ functional errors, where $\alpha$ is a user-defined parameter.

Error resilience capability works cooperatively with precision constraint table to select data precision. First of all, error resilience capability determines the worst case error rate. Then, based on the precision constraint table, the most inaccurate data precision among all acceptable configurations is selected.

### 4.2.4 *Precision Prediction Calibration*

Sometimes the predicted precision is not good enough due to the incompleteness of data subset and the inaccuracy of error resilience capability estimation model, leading to endless membership vibrations for some of the samples and significant energy consumption, which appears at the final stage of clustering algorithm.

To avoid the above situation, we keep track of the membership changes within the most recent two iterations. Whenever similar membership changes are detected, we would upgrade the data precision to be the next more accurate one.

## 4.3 Experimental Results

### 4.3.1 *Experimental Setup*

Experiments are conducted on two widely used clustering algorithms: Gaussian mixture model-based clustering (GMM) and k-means clustering (which is the hard assign version

Table 1: Applications and Datasets

| Application | Dataset | Source | Dimension | Distance Metric |
|---|---|---|---|---|
| k-means | svmguide | [19] | 4 | Least square distance with $\ell_2$ norm |
| | codrna | [19] | 8 | |
| | shuttle | [19] | 9 | |
| GMM | image noise | [20] | 3 | Mahalanobis distance with $\ell_2$ norm |
| | irisflowers | [19] | 2 | |
| | fourclasses | [21] | 2 | |

Table 2: Mixture Model-based Clustering with ApproxMA

| Application | Datasets | Model Deviation | Testing Data Error Rate | Energy Savings |
|---|---|---|---|---|
| k-means | svmguide | 0.3 | 0 | 58.08% |
| | codrna | 1.22e-04 | 0 | 41.92% |
| | shuttle | 9.39e-05 | 0 | 56.17% |
| GMM | image noise | 4.34e-05 | 0.07% | 51.41% |
| | irisflowers | 1.01e-06 | 0 | 42.08% |
| | fourclasses | 4.51e-03 | 0.60% | 59.64% |

of mixture model-based clustering [11]) with various datasets. Detailed information is listed in Table 1, wherein column "Source" gives the download source of our used datasets, and column "Dimension" indicate the dimension of features. column "Distance Metric" shows the distance metrics for each applications, which will be used in our runtime precision controller.

We apply two metrics to illustrate approximation quality loss on the final results. The first one is named as *model deviation*, which reflects distance between the approximated model and the accurate one. Here the distance is a model-specific metric, and we define it as follows: for k-means, Deviation of Models is calculated with the average Euclidean squared distance between the approximate and the accurate centroids (denoted as $c_{app}$ and $c_{acc}$, respectively), which is calculated as:

$$D = \frac{1}{k}\sum_{i=1}^{k} d(c_{app,i}, c_{acc,i}),$$

wherein $k$ is the number of clusters; For clustering with Gaussian mixtures, "Deviation of Models" is calculated with the weighted sum of Bhattacharyya distances between approximate and accurate Gaussian mixture models, which is calculated as,

$$D = \sum_{i=1}^{k} weight_i \times d(model_{app,i}, model_{acc,i}).$$

$d(model_{app,i}, model_{acc,i})$ denotes the Bhattacharyya distance between $model_{app,i}$ and $model_{acc,i}$. And $weight_i$ is the weight of accurate model.

The second metric is named as *Testing Data Error Rate*, which means the percentage of testing data that are assigned to a wrong cluster with our calculated approximate model. It should be noted that testing data is different from the training data that is used to calculated the model.

### 4.3.2 Energy Benefits

In this work, we mainly consider the energy savings of the proposed ApproxMA solution for memory accesses and the values are obtained with CACTI [18]. Note that, there is slight increase of energy consumption due to data reorganization before computation, but it is ignored in our experiments due to its relatively small contribution.

Table 2 demonstrates the quality loss and energy benefit for k-means clustering and Gaussian mixture clustering, separately. We can observe from the results that our proposed technique is able to achieve significant energy savings (from 40% to 60%) when comparing with fully accurate off-chip memory accesses. At the same time, the resultant quality loss is almost negligible. One interesting phenomenon illustrated by dataset *svmguide* is that the model deviation is as much as 0.3 while there is no testing data error detected. This is because the clusters are well separated in this dataset so that labeling the testing samples become easy and error-tolerant. Therefore, we are able to gain the best energy savings in this case, which illustrates that our proposed technique is very powerful to capture error resilience capability featured by specific dataset.

### 4.3.3 Comparison with Static Precision Scaling

We also conduct comparison with static precision scaling that uses a pre-determined fixed bit-width for approximation.

Experimental results for k-means clustering are shown in Table 3. For all the datasets, ApproxMA obtains the most accurate result. "Model Deviation" of dynamic precision scaling is the smallest (9.39e-05 vs 6.13 and 49.18 in Table 3(c)). Correspondingly, "Testing Data Error Rate" is zero indicating that models built with dynamic precision assign all testing data correctly. At the same time, the energy savings are significant, and sometimes better than the case when the bit-width is set as 16. This is because, static precision scaling with small bit-width may cause increase of iterations or even no convergence of the algorithm and therefore consumes more energy.

Generally speaking, the output quality decreases along with the reduction of bit-widths with static precision scaling, and the quality itself is not guaranteed, as shown in the unacceptable results in Table 3(b) and Table 3(c), where nearly half of the testing data are clustered incorrectly when the bit-width is set to be 12. This is a serious problem for static precision scaling because it has no knowledge on the actual data set used in practice.

**Table 3: Comparison of Static and Dynamic Precision Scaling**

| | (a) svmguide | | | (b) shuttle | | | (c) cod-rna | | |
|---|---|---|---|---|---|---|---|---|---|
| Bitwidth | Model Deviation | Testing Data Error Rate | Energy Savings | Model Deviation | Testing Data Error Rate | Energy Savings | Model Deviation | Testing Data Error Rate | Energy Savings |
| 16 | 0.45 | 0.13% | 54.55% | 1.81e-03 | 0.03% | 50.00% | 6.13 | 4.11% | 74.44% |
| 12 | 6.73 | 1.49% | 69.32% | 0.086 | 44.04% | 53.84% | 49.18 | 39.95% | 94.17% |
| dynamic | 0.3 | 0 | 58.08% | 1.22e-04 | 0 | 41.92% | 9.39e-05 | 0 | 56.17% |

## 5. CONCLUSION

Precision scaling is an effective approximate computing technique to improve energy-efficiency. However, most existing works focus on approximation of on-chip computations without considering the energy consumption of off-chip memory accesses. Since emerging Recognition, Mining, and Synthesis applications typically involve huge amounts of data accesses, in this work, we design an approximate memory access framework for dynamic precision scaling, namely ApproxMA. We apply this framework to mixture model-based clustering as a case study. As the precision requirements can vary significantly during runtime, we also propose a lightweight runtime bit-width calculator by jointly considering runtime data precision constraints and application's error resilience capabilities. Experimental results with two widely-used mixture model-based clustering algorithms (k-means and clustering with Gaussian mixtures) demonstrate the efficacy of the proposed methodology.

## 6. ACKNOWLEDGEMENT

This work was supported in part by National Natural Science Foundation of China (NSFC)/Hong Kong Research Grants Council (RGC) Joint Research Scheme N_CUHK444/12, and in part by NSFC Key Project 61432017.

## 7. REFERENCES

[1] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, "Impact: imprecise adders for low-power approximate computing," in *Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design*, pp. 409–414, 2011.

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

[3] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *Proceedings of the 50th Annual Design Automation Conference*, p. 113, 2013.

[4] Q. Zhang, F. Yuan, R. Ye, and Q. Xu, "Approxit: An approximate computing framework for iterative methods," in *Proceedings of the 51st Annual Design Automation Conference*, pp. 97:1–97:6, 2014.

[5] Q. Zhang, T. Wang, Y. Tian, F. Yuan and Q. Xu, "ApproxANN: An Approximate Computing Framework for Artificial Neural Network," *Proceedings IEEE/ACM Design, Automation, and Test in Europe (DATE)*, to appear, 2015.

[6] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Quality programmable vector processors for approximate computing," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 1–12, 2013.

[7] O. Sarbishei and K. Radecka, "Analysis of precision for scaling the intermediate variables in fixed-point arithmetic circuits," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 739–745, 2010.

[8] S. Yoshizawa and Y. Miyanaga, "Tunable wordlength architecture for a low power wireless OFDM demodulator," *IEICE Transactions on fundamentals of electronics, communications and computer sciences*, vol. 89, no. 10, pp. 2866–2873, 2006.

[9] S. Lee and A. Gerstlauer, "Fine grain word length optimization for dynamic precision scaling in dsp systems," in *2013 IFIP/IEEE 21st International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 266–271, 2013.

[10] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, "On reconfiguration-oriented approximate adder design and its application," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 48–54, 2013.

[11] V. Melnykov, R. Maitra, "Finite mixture models and model-based clustering," *Statistics Surveys*, vol. 4, pp. 80–116, 2010.

[12] C.M. Bishop *et al.*, *Pattern recognition and machine learning*, vol. 1. springer New York, 2006.

[13] H.-N. Nguyen, D. Menard, and O. Sentieys, "Dynamic precision scaling for low power WCDMA receiver," in *IEEE International Symposium on Circuits and Systems*, pp. 205–208, 2009.

[14] Y. Lee, Y. Choi, S.-B. Ko, and M. H. Lee, "Performance analysis of bit-width reduced floating-point arithmetic units in FPGAs: a case study of neural network-based face detector," *EURASIP Journal on Embedded Systems*, vol. 2009, p. 4, 2009.

[15] N. Luehr, I. S. Ufimtsev, and T. J. Martínez, "Dynamic precision for electron repulsion integral evaluation on graphical processing units (GPUs)," *Journal of Chemical Theory and Computation*, vol. 7, no. 4, pp. 949–954, 2011.

[16] H. Sangki, "3D super-via for memory applications," in *Micro-Systems Packaging Initiative Packaging Workshop (WSPI)*, 2007.

[17] V. Sathish, M. J. Schulte, and N. S. Kim, "Lossless and lossy memory I/O link compression for improving performance of GPGPU workloads," in *Proceedings of ACM International Conference on Parallel Architectures and Compilation Techniques*, 2012.

[18] "CACTI," http://www.hpl.hp.com/research/cacti/.

[19] http://www.csie.ntu.edu.tw/ cjlin/libsvm/

[20] http://scikit-learn.org/stable/modules/classes.html#module-sklearn.datasets

[21] T. K. Ho and E. M. Kleinberg, "Building projectable classifiers of arbitrary complexity," in *Proceedings of the 13th IEEE International Conference on Pattern Recognition*, vol. 2, pp. 880–885, 1996.