Error Analysis of the Square Root Operation for the Purpose of Precision Tuning: a Case Study on K-means

Oumaima Matoussi, Yves Durand, Olivier Sentieys, Anca Molnos

Error Analysis of the Square Root Operation for the Purpose of Precision Tuning: a Case Study on K-means

Oumaima Matoussi, Yves Durand
CEA, LETI, Univ. Grenoble Alpes, France
name.surname@cea.fr

Olivier Sentieys
Inria, Univ. Rennes, France
olivier.sentieys@inria.fr

Anca Molnos
CEA, LETI, Univ. Grenoble Alpes, France
anca.molnos@cea.fr

***Abstract*—In this paper, we propose an analytical approach to study the impact of floating point (FLP) precision variation on the square root operation, in terms of computational accuracy and performance gain. We estimate the round-off error resulting from reduced precision. We also inspect the Newton Raphson algorithm used to approximate the square root in order to bound the error caused by algorithmic deviation. Consequently, the implementation of the square root can be optimized by fittingly adjusting its number of iterations with respect to any given FLP precision specification, without the need for long simulation times. We evaluate our error analysis of the square root operation as part of approximating a classic data clustering algorithm known as K-means, for the purpose of reducing its energy footprint. We compare the resulting inexact K-means to its exact counterpart, in the context of color quantization, in terms of energy gain and quality of the output. The experimental results show that energy savings could be achieved without penalizing the quality of the output (e.g., up to 41.87% of energy gain for an output quality, measured using structural similarity, within a range of [0.95,1]).**

***Index Terms*—approximate computing, error analysis, round-off error, algorithmic deviation, square root, Newton Raphson method, precision tuning, k-means, clustering, floating point**

I. Introduction

The ever increasing volume, diversity and high dimensionality of data goes hand in hand with the rapid growth of energy consumption of computer systems. Thus, improving the energy efficiency of computer systems that try to keep pace with the constant growth of information is a critical concern.

In many application domains that deal with huge amounts of data such as multimedia processing (images, audio, video, etc.), data mining and machine learning, computations can be tolerant to some degree of error without critical degradation in the quality of the output. For example, a small image quality loss due to some modification of the color of a group of pixels can be hardly noticed by the user due to the limited capabilities of human perception. In such classes of applications, output accuracy could be traded for energy reduction.

Performance gains can be achieved at the application level using software techniques, thanks to a relatively new computing paradigm called *approximate computing* [1]. Introducing inexactness in computations may lead to energy scaling with little to no loss in accuracy. A way to introduce inexactness in an application is by precision reduction of FLP variables and computations [2], [3]. Finding the sweet spot between the numerical accuracy and the performance of the application (e.g. energy consumption) by refining the bit-width of the FLP variables is known as *precision tuning*.

A loss in computational accuracy is inevitable due to reduced precision. However, to be able to determine the optimal precision (i.e. bit-width) of FLP variables that minimizes energy cost with the least impact on computational accuracy, two approaches can be considered: FLP simulation and analytical techniques to track round-off error (i.e. error introduced due to limited precision). FLP simulation consists in using libraries (e.g. MPFR) that allow the definition of FLP variables with adjustable bit-widths and provide the appropriate arithmetic operations. Nonetheless, the search space of the optimal bit-width can be very large, in that every change in the format of a FLP variable requires a new simulation, which can be very time consuming.

Analytical approaches, on the other hand, try to determine a mathematical formula that models the impact of quantization error on the accuracy of the output [4], [5]. Once this formula is established, it can be applied to any application with different FLP formats. Unfortunately, this only works with smooth operations (arithmetic operations like addition, subtraction, multiplication, division, etc.). So, analytical approaches help save the time spent on tuning FLP precision but fail to estimate the computational error in the presence of non-smooth operations. In this case, simulation becomes inescapable. Thus, we advocate through our work the combination of simulation and analytical techniques.

A plethora of work using simulation-based approaches for the purpose of FLP precision tuning was propounded [3], [6], [7], etc. As for analytical approaches, efforts were made in the context of fixed-point word-length optimization [4], [8] and mainly focused on common arithmetic operations like addition and multiplication. Square root operation is significantly more costly, in terms of energy consumption, than addition or multiplication and requires several iterative cycles to complete. Moreover, unlike simple arithmetic operations, the square root is usually implemented using the *Newton Raphson* method, which is an approximation of the operation itself, and adds another type of error referred to as *algorithmic deviation* besides the round-off error. Consequently, error analysis of a square

root operation is not a straightforward task and entails a two-fold method.

The main contribution of this paper consists in conducting static error analysis of the square root operation and estimating a bound on the errors caused by limited precision and algorithmic deviation. Based on the formally derived error bound, the number of iterations of the square root operation is automatically adjusted for each precision. Therefore, the implementation of the square root (in terms of number of iterations) can be optimized with respect to any given FLP precision specification without the need for long simulation times.

In addition to the stand-alone analysis of the square root operation and for a more complete evaluation of our proposition, the second contribution of this paper consists in evaluating our analytical approach by studying the square root operation as part of a whole application. Our choice fell on a data clustering algorithm called K-means a.k.a. Lloyd's algorithm [9]. It is a type of unsupervized learning algorithm used to cluster a set of unlabeled data into k clusters based on data feature similarity [10]. The similarity is typically determined using the Euclidean distance measure, which involves square root operations. The distance function is the mainstay of the majority of clustering and classification algorithms, which will help us highlight the impact of square root optimization on the performance of the application as a whole.

It should be noted that the proposed analytical approach for the square root operation is independent of the application itself. Any clustering algorithm or any other application (e.g. digital signal and image processing, 3D graphics, spectrum analysis, wireless communications, etc.), for that matter, where square root computations can be found, would also benefit from our error analysis method. The importance of our contribution lies within the integration of our analytical approach, which is aimed for a smooth operation, namely the square root, in an application that also contains non-smooth operations. Hence, the combined effort of simulation and analytical results is needed for precision tuning. We quantify the efficiency of the proposed approach by measuring the quality degradation of K-means' output as well as its energy gain.

The rest of this paper is organized as follows. After overviewing contributions dealing with approximate computing in Section II, we detail our error analysis approach of the square root operation in Section III. We explain the process of sensitivity analysis of K-means, based on our square root analysis results and variable-precision FLP simulation in Section IV. We validate the efficiency of approximate K-means by measuring both SSIM and energy gain and we also discuss the experimental results in Section V, before concluding the paper in Section VI.

II. Related Work

Precision tuning is a research direction that has received significant attention and is motivated by the fact that FLP operations contribute to the energy footprint of an application.

In [6], a framework called ASAC using statistical methods was proposed. This framework helps discover approximable and non-approximable program parts through sensitivity analysis. The main idea is to perturb program variables and check the resultant output against a correct output, i.e. one fulfilling an acceptable QoS threshold.

Precimonious [7], a dynamic program analysis tool, was proposed to assist developers in choosing the lowest precision that satisfies accuracy and performance constraints. Given a set of FLP variables and their possible types, a search based on the delta-debugging algorithm is performed. The search outputs a type configuration that maps each variable to a type. However, Precimonious requires a representative set of program inputs provided by the user. If the same type configuration is applied on a much worse conditioned input then no guarantees can be made.

The method proposed in [2] is not limited to determining the best mix of single or double precision in a program as in [7], but it computes the precision of FLP variables down to the bit level of the mantissa. A heuristic precision tuning algorithm based on a binary search is performed in order to find the smallest precision possible for each variable while keeping the output error within some user-given bound. The results show that the speedup gained by tuning some programs may be diminished or even eliminated because of the overhead due to data type conversions performed by the compiler.

In [3], hardware implementations of fixed-point and FLP arithmetic operators are compared in terms of area, delay and energy. A custom FLP library called *ct_float* was devised to vary the bit width of the variables. The authors concluded that FLP operators provide a better energy/accuracy tradeoff for small bitwidths but important area, delay and energy overhead for larger bitwidths, compared to fixed-point operators.

The common denominator in the majority of FLP precision tuning approaches is that they rely on varying the precision of the different FLP variables in an application (usually with the help of a multiple precision FLP library) and testing different combinations before deciding on the optimal precision. This process is prohibitively expensive, time wise. Furthermore, when big applications with a huge set of input is under optimization, the search space for optimal precision can be enormous and it might be impossible to cover fully.

Analytical approaches are faster than simulation because once a mathematical expression that models (or bounds) the round-off error is determined, it can be applied to different precisions. So, it is a one-time effort whose results can be applied to multiple FLP formats. Efforts have been made to find analytical models in the context of accuracy evaluation of fixed-point systems. A series of propositions of analytical methods to measure the output noise of signal processing systems were presented in [4], which dealt with smooth operations, then in [5], which catered for decision operators and recently in [8], where a hybrid approach based on both simulation and analytical results was highlighted.

A parametric error analysis of a version of Goldschmidt's square root algorithm that uses directed rounding implemen-

tations, was proposed in [11]. The error analysis is based on relative errors of intermediate computations. The proposed error formulae were intended to help determine the optimal hardware implementation (i.e. multiplier dimensions). However, their analysis was not demonstrated on a real architecture.

In our work, we study analytically the square root operation, which is the backbone of many algorithms such as clustering, data mining and signal processing algorithms, with respect to FLP precision tuning. Moreover, we quantify the efficiency of the proposed error bound in the context of K-means by comparing the QoS of the inexact version of K-means to its exact counterpart. We also measure relative energy gains using energy costs obtained from [3], [12] and profiling information.

III. Analysis of the Square Root Operation

First, to eliminate any confusion, it should be noted that the terms accuracy and precision are not used interchangeably in this paper. Nearly all processors and programming languages support FLP numbers, which are defined in the IEEE-754 normalization. A FLP number f is represented by an exponent e, a mantissa m and a sign bit s: $f = (-1)^s \times m \times 2^e$. We designate by precision the number of bits used to represent the mantissa of a FLP variable, whereas we refer by accuracy to the quality of the result, i.e., the degree of error in the output, compared to a reference or a *golden* output.

The square root operation is usually approximated with the *Newton Raphson* method. To compute $y = \sqrt{a}$, $a > 0$, the Newton Raphson method starts with an initial guess (i.e. initial seed value) $y_0 > 0$. The initial guess is then refined by iterating over:

$$y_{n+1} = \frac{1}{2}(y_n + \frac{a}{y_n})$$

However, this conventional iteration for the square root computation is not frequently used because it entails a division at each step. Since division is generally much slower than multiplication, the square root reciprocal is usually advocated. The square root reciprocal converges to $\frac{1}{\sqrt{a}}$ and iterates over: $x_{n+1} = \frac{x_n}{2} \times (3 - a \times x_n^2)$.
In order to get $\sqrt{a}$, the result is multiplied by a.

To optimize the implementation of the Newton Raphson method, we aim at adjusting the number of its iterations according to the selected precision, without jeopardizing the accuracy of the result. To do so, we perform error analysis to statically determine at which iteration it is preferable to stop the computations, for a specific precision p, without causing further error. The two main causes of error that we investigated are:

- the round-off error caused by FLP representation and FLP operations,
- and the systematic error (a.k.a. unavoidable error or algorithmic deviation), which results from the Newton Raphson approximation itself.

A. Bounding the Round-off Error

Let $fl(x+y) = (x+y)(1+\epsilon_{add})$ and $fl(x \times y) = (x \times y)(1+\epsilon_{mul})$, be approximations of the exact mathematical operations $+$ and $\times$ respectively, $\forall x, y \in \mathbb{R}$, such that $|\epsilon_{add,mul}| \leq \epsilon_m$. ϵ_m, called *machine epsilon* or the *unit roundoff*, is defined as the smallest number such that $1 + \epsilon_m > 1$ [13]. $\epsilon_m = \beta^{-p}$, where β is the base and p is the number of bits used for the magnitude of the mantissa of a FLP number represented as $r_0.r_1r_2...r_{p-1} \times \beta^e$. We assume that $\epsilon_{add} = \epsilon_{mul} = \epsilon$ and that multiplication by $\frac{1}{2}$ does not incur round-off error.
$\forall x_i \in \mathbb{R}$, let $\hat{x}_i = fl(x_i)$ be the FLP representation of x_i; $i = 1..n+1$. Machine epsilon is an upper bound on the relative error in representing a FLP number: $\frac{|\hat{x}_i - x_i|}{|x_i|} \leq \epsilon_m$.

We would like to bound the round-off error of the Newton Raphson reciprocal. To do so, we aim at expressing $\hat{x}_{n+1}$ as: $\hat{x}_{n+1} \leq x_{n+1} \times (1 + \delta)$, where δ is the round-off error.

$$\hat{x}_{n+1} = fl(x_{n+1}) = fl\Big(\frac{\hat{x}_n}{2} \times fl\Big(3 - fl\big(a \times fl(\hat{x}_n^2)\big)\Big)\Big)$$

$$\hat{x}_{n+1} = \frac{\hat{x}_n}{2} \times \Big(3 - a \times \hat{x}_n^2(1+\epsilon)^2\Big)(1+\epsilon)^2 \tag{1}$$

Let $e(\hat{x}_n) = 3 - a \times \hat{x}_n^2(1+\epsilon)^2$.

$$|e(\hat{x}_n)| \leq |3 - a \times \hat{x}_n^2| + |2a \times \hat{x}_n^2 \times \epsilon| \tag{2}$$

We have:

$$\sqrt{a} \times x_n < 1 \tag{3}$$

Proof. $x_{n+1}\sqrt{a} - 1 = -(1 + x_n\frac{1}{2}\sqrt{a})(x_n\sqrt{a} - 1)^2 < 0$ □

Based on Equation 3 we can conclude that

$$\epsilon < \frac{(3 - a \times x_n^2) \times \epsilon}{2}. \tag{4}$$

Then, combining Equations 2 and 4 gives:

$$|e(x_n)| < (3 - a \times x_n^2) \times (1 + \epsilon), \tag{5}$$

and combining Equations 1 and 5 gives

$$\hat{x}_{n+1} < \frac{\hat{x}_n}{2} \times (3 - a \times \hat{x}_n^2) \times (1+\epsilon)^3.$$

Disregarding $O(\epsilon^n)$ terms, $n > 1$, in the previous inequality [13], we obtain

$$\hat{x}_{n+1} < \frac{\hat{x}_n}{2} \times (3 - a \times \hat{x}_n^2) \times (1 + 3\epsilon), \tag{6}$$

from which we can conclude that the round-off error is 3ϵ per iteration. Note that $\hat{x}_{n+1}$ converges towards $\frac{1}{\sqrt{a}}$ and that we want to bound the error of $\hat{y}_{n+1}$, which converges towards $\sqrt{a}$. So, only the last iteration should be multiplied by a:

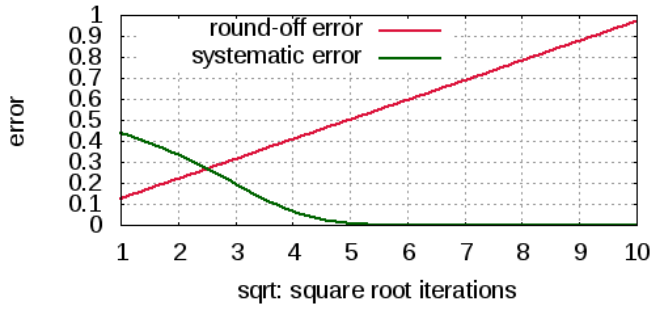
$$x_{n+1} \times a = y_{n+1} \Longrightarrow fl(x_{n+1} \times a) = fl(y_{n+1})$$
$$\Longrightarrow \hat{x}_{n+1} \times a \times (1+\epsilon) = \hat{y}_{n+1}.$$

Consequently, the overall round-off error is $\delta \leq n \times 3\epsilon + \epsilon$.

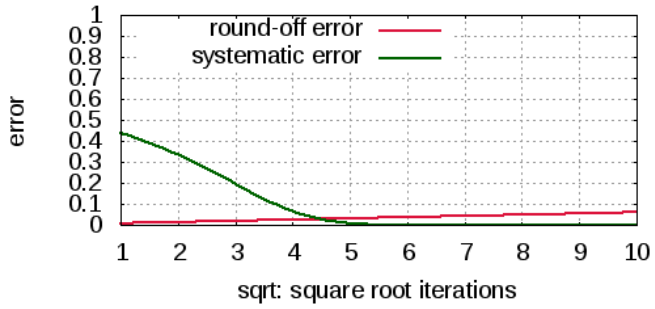
B. Bounding the Systematic Error

The absolute systematic error in computing $\frac{1}{\sqrt{a}}$ is

$$|x_i - \frac{1}{\sqrt{a}}|.$$



(a) p=4



(b) p=9

Fig. 1: Systematic and round-off errors for different precision values and iteration values of *Newton Raphson*

The relative systematic error in computing $\frac{1}{\sqrt{a}}$ is

$$\frac{|x_i - \frac{1}{\sqrt{a}}|}{\frac{1}{\sqrt{a}}} = |\sqrt{a}x_i - 1|.$$

To find an upper bound for the systematic error, we start with a reasonable initial value defined as $\frac{1}{2} \times \frac{1}{\sqrt{a}} \leq x_0 \leq \frac{3}{2} \times \frac{1}{\sqrt{a}} \Longrightarrow 1 + 0.5 \times \sqrt{a} \times x_0 \leq \frac{7}{4}$ and $\sqrt{a} \times x_0 - 1 \leq \frac{1}{2}$.
So, according to the proof of Equation 3:
$|\sqrt{a} \times x_1 - 1| \leq \frac{4}{7} \times |\frac{7}{4} \times (\sqrt{a} \times x_0 - 1)|^2$.
Since $\frac{7}{4} \times (\sqrt{a} \times x_0 - 1) \leq \frac{7}{8}$ then, $|\sqrt{a} \times x_1 - 1| \leq \frac{1}{2} \times \frac{7}{8}$,
It is safe to conclude that the systematic error in computing $\frac{1}{\sqrt{a}}$ at iteration $i = 0..n$ is

$$|\sqrt{a} \times x_i - 1| \leq \frac{1}{2} \times (\frac{7}{8})^{2^i - 1}. \quad (7)$$

The relative systematic error for $\frac{1}{\sqrt{a}}$ is the same as the one for $\sqrt{a}$: $\frac{|y_i - \sqrt{a}|}{\sqrt{a}} = \frac{|ax_i - \sqrt{a}|}{\sqrt{a}} = |\sqrt{a}x_i - 1|$.

Both the systematic and round-off errors are plotted for different precision values (p) in Fig. 1. For space reason, only two of the resulting graphs are illustrated. The straight line corresponds to the round-off error, which increases proportionally to the number of iterations of the *Newton Raphson* method, whereas the curve designates the systematic error, which declines as the number of iterations grows.

The rise of the round-off error is remarkably rapid at lower precision values (e.g. Fig. 1(a)) but it slows down at higher precision (e.g. Fig. 1(b)).

The systematic error, on the other hand, is independent of the precision. It stabilizes at zero around the sixth iteration. This means that, disregarding the round-off error (round-off error equal to 0), six *Newton Raphson* iterations are sufficient in producing accurate (i.e. error=0) square root approximation. However, when the round-off error is in the picture, which is usually the case, the higher the number of iterations, the higher the round-off error is. Although at six $sqrt$ iterations the systematic error is 0, the round-off error is 0.6 for $p = 4$ and 0.05 for $p = 9$.

The round-off error accumulates as the number of iterations rises, which counteracts the decline of the unavoidable error. That is why, the round-off error should not exceed the systematic error. Thus, the intersection between the two lines is indicative of the *optimal* number of iterations at which we have a *good enough* result.

IV. Application to K-means

So far, we studied the square root operation individually, in the context of error analysis for the purpose of precision tuning. It would be interesting to apply our study of the square root in a fullblown application that encompasses both smooth and non-smooth operations. We chose a clustering algorithm called K-means that uses the square root operation in the computation of the Euclidean distance function.

In this section, we start by describing K-means and its use in color quantization. Then, we present precision tuninig of K-means that involves both FLP simulation and our analytical results.

A. Color Quantization Using K-Means

The basic principle of the clustering problem is as follows. A data set $X = (x_1, x_2, ..., x_n)$ is composed of data elements, where each data element x_i is a d-dimensional vector. This data set is partitioned into k clusters of *similar* data points [14].

In the particular case of color quantization, one of K-means' applications, an image represented as an array of size $N \times d$ where N is the number of pixels in the image and d is the color space (referred to as data dimension or data features), is represented with a smaller number of colors. Most commonly, the space is 3-dimensional ($d = 3$) and the coordinates encode the color. For example, *RGB* is a color space usually encoded as a 3-tuple of 8 bits each. The value of each dimension is within the range of $[0, 255]$.

The flowchart in Fig. 2 gives an overview of the different steps of the clustering process in the context of color quantization with K-means.

The goal of the K-means clustering algorithm is to generate a compressed image out of the original image. To this end, a *palette*, i.e. a set of centroids $C = (c_1, c_2, ..., c_k)$, is firstly chosen by selecting data elements (i.e. colors or pixels) that best represent the original image, for each cluster. In other words, the number of centroids is the number of colors that the palette is made of. Then, each data element is mapped to the *closest*

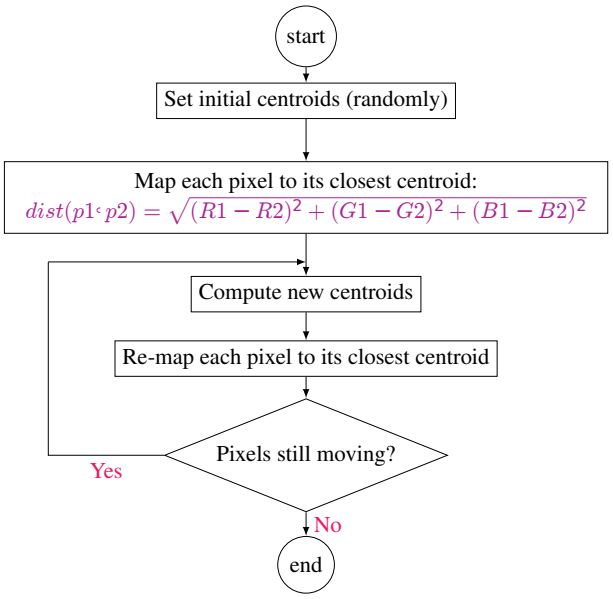


Fig. 2: Flowchart of the K-means algorithm

color in the palette based on a distance measure $\|x\|_m = (\sum_{i=1}^{n} |x_i|^m)^{1/m}$ (e.g. $m = 1$ for the Manhattan distance, $m = 2$ for the Euclidean distance) [15]. This results in a preliminary classification. Following this initial classification, k new centroids are re-computed as barycenters of the previously generated clusters and data elements are re-assigned to the new centroids based on the chosen distance measure. The algorithm is based on an iterative refinement technique and it iterates until centroids do not move anymore (i.e., no data elements change clusters), a maximum number of iterations is reached or the distance measure is minimized, as illustrated in Fig. 2. To quantify the difference between two pixels, a simpler distance function known as the Manhattan distance (sum of absolute differences) is usually used. However the optimal distance, in terms of minimizing within-class variance and producing higher-quality clusters, and more complicated, in terms of computational complexity, is the Euclidean distance (sum of the squared distances) [16]. Unlike other studies on approximate K-means, we use the optimal distance metric:

$$dist(p1, p2) = \sqrt{(R1-R2)^2+(G1-G2)^2+(B1-B2)^2}.$$

B. Precision Tuning Using a Multiple-Precision Floating Point Library

We focus our precision analysis on the Euclidean distance function, which is the kernel of the K-means algorithm as it is in charge of computing the distance between a given data point and a cluster centroid. These distance values are pivotal in the correct assignment of pixels to their closest cluster and thus in providing a correct classification. We are going to include approximation in the FLP computations of the Euclidean distance function and leverage the results from our analysis in Section III to account for the impact of FLP precision variation on the square root operation.

Introducing inexactness by reducing the number of bits of the mantissa can sometimes provide results of the same accuracy but with better energy efficiency than the exact version of the program. To this aim, we studied the sensitivity of K-means by arbitrarily varying the number of bits of the mantissa (2-23 bits) and observing the repercussions on the output.

As was established in our analytical study of the square root operation (Section III), the number of bits of the mantissa has an impact on the number of square root iterations. So, to study the sensitivity of K-means, we re-wrote K-means' Euclidean distance function using the MPFR library, which is a smooth extension of the IEEE-754 standard where any FLP number can have its own precision [17], and we assigned to each precision its corresponding number of square root iterations according to the analysis in Section III. Our analytical results come in handy in the precision tuning of K-means in that, we do not have to vary the number of square root iterations for each precision simulation in order to determine the optimal pair $(p, sqrt)$. The optimal number of square root iterations is statically determined for each precision (Section III). For example, for a precision $p = 4$, the FLP variables in K-means' Euclidean distance function are transformed into MPFR variables ($mpfr\_t\ var$), their precision is set to 4 ($mpfr\_init2(var, 4)$) and the number of iterations in the Newton Raphson method implementation is set directly to $sqrt = 2$.

In addition to the FLP variables, we tracked the operations that use these variables and changed them into MPFR operations. The Euclidean distance function encompasses multiplication, addition, subtraction and square root operations.

Listing 1: MPFR operations

```
1 mpfr_sub(r,r,r1,MPFR_RNDN); /* r=r-r1; */
2 mpfr_mul(r,r,r,MPFR_RNDN); /* r=r*r; */
3 mpfr_add(r,r,r2,MPFR_RNDN); /* r=r+r2; */
4 mpfr_sqrt(r_tmp,r,MPFR_RNDN); /* r_tmp=sqrt(r) */
```

Listing 1 showcases the transformation of different operations into MPFR operations. We implemented two inexact versions of the Euclidean distance function. We called these versions fused and unfused. In the first, the sub, mul and add operations are computed with full precision and the result is rounded once to N significant bits and then passed to the reduced-precision square root function. In the latter, all the operations are performed with reduced-precision, which means that the value is rounded four times before yielding the final result. We examine both fused and unfused versions in Section V to determine which version is more beneficial to implement.

The quality of the result as well as the energy consumption depend not only on the precision p of the FLP variables but also on other parameters such as the number of clusters k and the number of iterations n needed for the clustering process to converge. More clusters means that the compressed image will contain more colors, which is in favor of a better quality. With more iterations, there is a better chance for K-means to converge, which also contributes to the quality of the output. So, we investigated precision variation and its impact on the

square root iterations over a range of different values of (k, n) pairs. Thus, we also varied the (k, n) parameters along with the precision and quantified the impact on the output in order to discern the best configuration that yields the desired energy-QoS tradeoff.

V. Experimental Results

To determine whether a configuration (k, n) for a given precision p is favorable, we transform the original source code of K-means [18] using the configuration in question, compile it to an ARM binary, execute it while using energy and instruction counters, and check for two criteria: energy consumption and QoS.

A. Experimental Setup

In order to quantify the impact of approximation on energy consumption, we make use of an energy model based on energy measurements of ARM Cortex-A7 instructions at a frequency of 500 MHz and without dependencies between instructions [12]. We also make use of profile information, i.e., instruction types (e.g., add, mul, div) and the number of execution of each type. ARM's Cortex-A7 core is a dual issue in-order core that consists of one load/store, one multiply, one FLP and two integer units. A thorough characterization of the ARM Cortex-A7 instruction set with energy metrics for every instruction type could be found in [12].

The energy values of integer and FLP instructions are normalized with reference to integer multiplication, i.e., integer multiplication is used as a unit of measure throughout the experiments. Normalized energy values of 32-bit integer and FLP instructions are listed in Table I. These normalized energy values are used to compute the energy consumption of exact K-means, which is considered as a touchstone. As for the energy values of variable precision (2-16 bits) operations, namely addition, multiplication and division, they are computed according to [3] and shown in Fig. 3. It is noticeable from Fig. 3 that energy consumption is influenced by the number of bits of the mantissa; the higher the precision the higher the energy cost.

The overall energy consumption is computed by combining the profile information and the energy values

$$E_{total} = \sum_{i=1}^{\#types} op_i \times e_i,$$

where op_i is the number of operations of type i and e_i is the energy consumed per operation of type i (normalized to $muli$). We estimate the energy savings by comparing the energy consumption obtained by executing the exact version of K-means (i.e. with the highest precision $p = 23$) to the approximated version.

The quality of the result is measured using the SSIM index. SSIM is a perception-based metric that compares two images by incorporating a number of terms including contrast, luminance and structural information. The degradation of the image quality, due to compression with inexact K-means (reduced precision and/or parameter variation), is determined with respect to the compressed image with the highest precision.

TABLE I: Energy values (normalized to $muli$) of a sample of integer and 32-bit FLP instructions

Instr	addi	muli	si	li	addf	mulf	sf	lf	divf
Energy	1.02	1	2.41	1.79	1.16	1.16	2.34	1.92	7.80

instri:integer instruction, instrf:FLP instruction

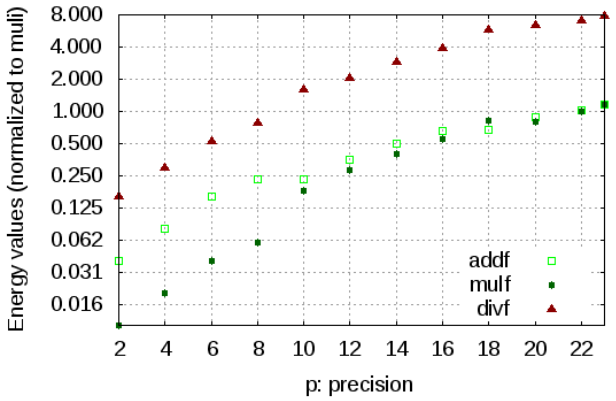


Fig. 3: Normalized energy values for different precisions

B. Results

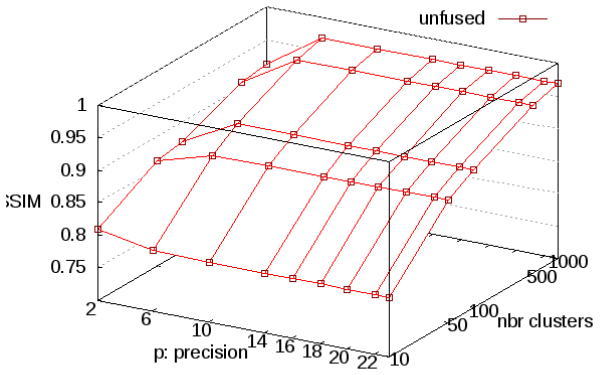
In this section, we study the impact of the precision and square root iterations (p, $sqrt$) on the quality of the output for various n (number of iterations needed for K-means to converge) and k (number of clusters) configurations. To do so, we start by jointly varying the precision bits p and the number of K-means iterations n, while fixing the number of clusters to $k = 50$, for both unfused and fused operations.

Then, we vary the number of clusters k and the precision p while fixing the number of K-means iterations to $n = 10$. The SSIM values are averaged over 10 different RGB images chosen from [19]. An SSIM value is within the interval $[0, 1]$, 1 designating the best image quality.

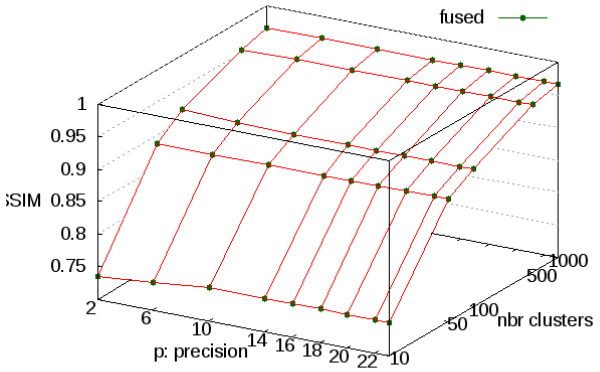
The first set of experiments that we conducted, i.e., SSIM(p, n) shows that the number of iterations n has little to no impact on SSIM for different precisions and for both the fused and unfused cases. This is true for other k values as well $(10, 100, 500, 1000)$. Accordingly, the number of iterations n is fixed to 10 for the rest of the experiments.

However, by varying the precision with the number of clusters while fixing n to 10, a noticeable change in SSIM is observed and the results are reported in Fig. 4. For instance, in case of unfused operations (Fig. 4(a)), SSIM increased from 0.791 with $k = 10$ to 0.969 with $k = 1000$, for $p = 10$. Regarding the fused operations' case (Fig. 4(b)), the results are almost identical to the unfused case except for $k = 10$, where the unfused version yielded better SSIM values for the different tested precisions, and for $p = 2$ and $k > 10$, where the fused version yielded better SSIM values.

Based on our experiments (SSIM(p, k) in Fig. 4 and SSIM(p, n)), fused and unfused operations generate similar



(a) unfused operations



(b) fused operations

Fig. 4: Impact of precision variation and number of clusters on SSIM: for (a) unfused operations and (b) fused operations

SSIM values for the majority of the configurations. Taking into consideration the additional requirements that come with fused operations (i.e., dedicated hardware), which is not always supported by all architectures, as well as a compiler that allows the program to make use of the fused operations, we deem it too big an effort for an insignificant quality enhancement. Consequently, we advocate the use of unfused operations in the context of reduced-precision K-means.

It is also worth mentioning that we conducted the experiments from $p = 2$ to $p = 23$, but no noticeable change was detected starting from $p = 16$, as can be observed in Fig. 4. Thus, the rest of the graphs presented in this paper stop at $p = 16$.

To conclude whether approximating K-means is worthwhile, we also measured the energy gains. Fig. 5 displays the number of executed instructions of the Euclidean distance function per intruction type: integer arithmetic, FLP arithmetic, load/store (both FLP and integer) and other instructions (e.g. branch,

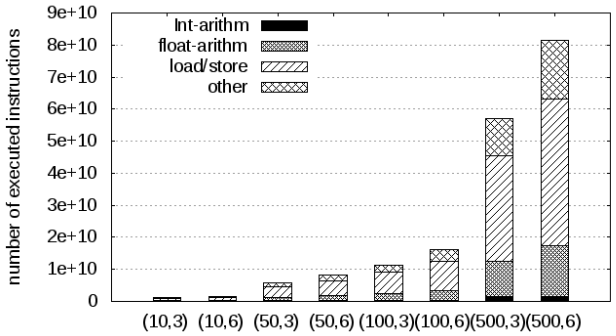


Fig. 5: Executed instructions breakdown for exact K-means $(p = 23)$ with different $(k, sqrt)$ configurations

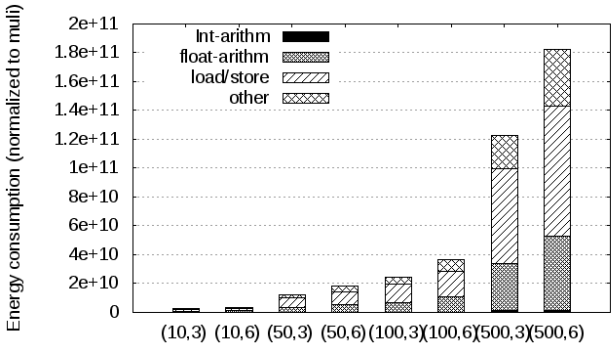


Fig. 6: Energy consumption breakdown for exact K-means $(p = 23)$ with different $(k, sqrt)$ configurations

comparison, conversion). The values are generated by applying exact K-means ($p = 23$) while varying the number of clusters and the square root iterations. For the different configurations of $(k, sqrt)$, the FLP arithmetic operations contribute with around 19% of the total number of executed instructions compared to integer arithmetic operations, which represent only 2% when $sqrt = 6$ and almost 3% when $sqrt = 3$.

Fig. 6 shows the energy consumption breakdown for exact K-means with different $(k, sqrt)$ configurations. It is clear that the number of clusters k as well as the number of square root iterations $sqrt$ have a significant impact on the energy footprint of the program. For instance, for the same number of clusters $k = 50$, energy consumption increases by approximately $6{\times}10^9$ from $sqrt = 3$ to $sqrt = 6$.

The percentages of energy savings are plotted against p for different $(k, sqrt)$ configurations in Fig. 7. Each precision is attributed its optimal number of square root iterations according to our analytical study (Section III). For example, for $p = 4$, $sqrt = 2$. The graph indicates that there are energy gains for the different $(p, k, sqrt)$ configurations but with different magnitudes. The energy gain of each configuration $(p, k, sqrt)$ is computed with respect to the reference configuration $(p = 23, k = 100, sqrt = 6)$. The smallest gain is observed at $p = 16$ and $(k = 100, sqrt = 6)$ with a value of 14.09%. The biggest gain is at $p = 2$ and $(k = 10, sqrt = 1)$ with a value of 96.57%. To be able to determine the best energy-QoS tradeoff

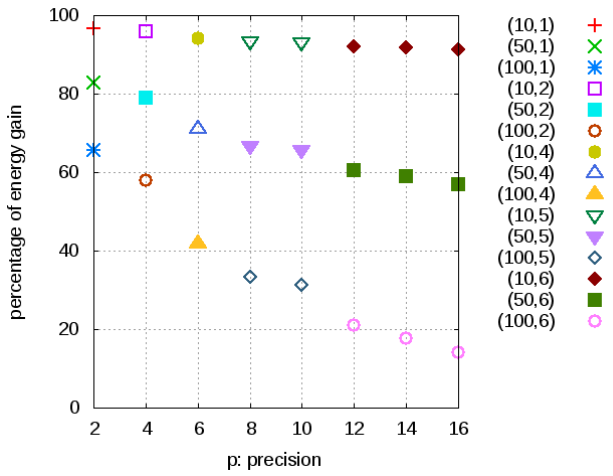


Fig. 7: Percentage of energy gain of different precisions p and with different $(k, sqrt)$ configurations

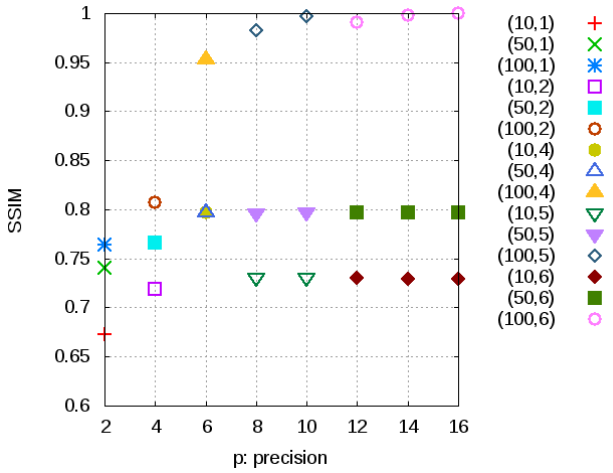


Fig. 8: SSIM of different precisions p and with different $(k, sqrt)$ configurations

for each precision, SSIM is plotted against p in Fig. 8 with the same $(k, sqrt)$ configurations as in Fig. 7. The same reference $(p = 23, k = 100, sqrt = 6)$ also serves as a baseline. The best quality is reached at $p = 16$ and $(k = 100, sqrt = 6)$ with an SSIM value of 1. The worst quality is noticed at $p = 2$ and $(k = 10, sqrt = 1)$ with an SSIM value of 0.67.

Assuming that an output image with an SSIM value lying within the range of $[0.95, 1]$ is considered a *good* quality image, then configuring K-means with a 6-bit mantissa, 100 clusters and 4 square root iterations is sufficient in producing such an image while saving 41.87% of energy.

VI. Conclusion

This paper focused on optimizing the implementation of the square root operation. An analytical examination of the *Newton Raphson* approximation of the square root showed that the number of square root iterations is influenced by the precision bits. Consequently, we associated to each precision its optimal number of *Newton Raphson* iterations. For a well-rounded evaluation of our proposition, we aimed at finding opportunities to reduce energy consumption of a classic clustering algorithm called K-means by varying the precision of its FLP variables and adjusting the number of square root iterations of its distance function.

The various approximated versions of K-means were compared to the *exact* version in terms of QoS, measured with SSIM, and relative energy gain. The obtained results can serve as a guideline in choosing the *best* configuration of precision bits, number of clusters and square root iterations $(p, k, sqrt)$, i.e., one that yields the most energy gains, for a desired QoS (e.g., for an SSIM within $[0.95, 1]$, an energy gain of 41.87% is achieved with a $(6, 100, 4)$ configuration).

References

[1] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," *IEEE Design Test*, Feb 2016.

[2] N. M. Ho, E. Manogaran, W. F. Wong, and A. Anoosheh, "Efficient floating point precision tuning for approximate computing," in *ASP-DAC*, Jan 2017.

[3] B. Barrois and O. Sentieys, "Customizing Fixed-Point and Floating-Point Arithmetic - A Case Study in K-Means Clustering," in *IEEE International Workshop on Signal Processing Systems*, Oct. 2017.

[4] R. Rocher, D. Menard, P. Scalart, and O. Sentieys, "Analytical approach for numerical accuracy estimation of fixed-point systems based on smooth operations," *IEEE Transactions on Circuits and Systems*, pp. 2326–2339, Oct 2012.

[5] K. Parashar, R. Rocher, D. Menard, and O. Sentieys, "Analytical approach for analyzing quantization noise effects on decision operators," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, March 2010, pp. 1554–1557.

[6] P. Roy, R. Ray, C. Wang, and W. F. Wong, "Asac: Automatic sensitivity analysis for approximate computing," in *SIGPLAN/SIGBED LCTES*. New York, NY, USA: ACM, 2014.

[7] C. Rubio-González, C. Nguyen, H. D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey, C. Iancu, and D. Hough, "Precimonious: Tuning assistant for floating-point precision," in *SC*, Nov 2013.

[8] K. N. Parashar, D. Menard, and O. Sentieys, "Accelerated performance evaluation of fixed-point systems with un-smooth operations," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 599–612, April 2014.

[9] G. Hamerly and J. Drake, *Accelerating Lloyd's Algorithm for k-Means Clustering*. Springer International Publishing, 2015.

[10] J. Silva, E. Faria, R. Barros, E. Hruschka, A. de Carvalho, and J. Gama, "Data stream clustering: A survey," 2014.

[11] P. Seidel, "A parametric error analysis of goldschmidt's square-root algorithm," in *ACSSC*, Nov 2015, pp. 727–731.

[12] E. Vasilakis, "An instruction level energy characterization of arm processors," https://www.ics.forth.gr/carv/greenvm/files/tr450.pdf, CARV Laboratory ICS FORTH, Tech. Rep., 2015, 7/8/2018.

[13] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, 2002.

[14] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM Comput. Surv.*, Sep. 1999.

[15] P. Sinha and R. Russell, "A perceptually based comparison of image similarity metrics," *Perception*, 2011.

[16] M. Leeser, J. Theiler, M. Estlick, and J. J. Szymanski, "Design tradeoffs in a hardware implementation of the k-means clustering algorithm," in *IEEE SAM SP Workshop*, 2000.

[17] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann, "Mpfr: A multiple-precision binary floating-point library with correct rounding," *ACM Trans. Math. Softw.*, no. 2, Jun. 2007.

[18] A. Yazdanbakhsh, D. Mahajan, P. Lotfi-Kamran, and H. Esmaeilzadeh, "Axbench : A benchmark suite for approximate computing across the system stack," 2016.

[19] "Uncompressed rgb images," https://bitbucket.org/act-lab/axbench/src, accessed: 06/08/2018.