Customizing Fixed-Point and Floating-Point Arithmetic - A Case Study in K-Means Clustering

Benjamin Barrois, Olivier Sentieys

Customizing Fixed-Point and Floating-Point Arithmetic – A Case Study in K-Means Clustering

Benjamin Barrois
University of Rennes 1 – IRISA/INRIA
Email: benjamin.barrois@irisa.fr

Olivier Sentieys
INRIA – University of Rennes 1
Email: olivier.sentieys@inria.fr

***Abstract*—This paper presents a comparison between custom fixed-point (FxP) and floating-point (FlP) arithmetic, applied to bidimensional K-means clustering algorithm. After a discussion on the K-means clustering algorithm and arithmetic characteristics, hardware implementations of FxP and FlP arithmetic operators are compared in terms of area, delay and energy, for different bitwidth, using the *ApxPerf2.0* framework. Finally, both are compared in the context of K-means clustering. The direct comparison shows the large difference between 8-to-16-bit FxP and FlP operators, FlP adders consuming 5-$12\times$ more energy than FxP adders, and multipliers 2-$10\times$ more. However, when applied to K-means clustering algorithm, the gap between FxP and FlP tightens. Indeed, the accuracy improvements brought by FlP make the computation more accurate and lead to an accuracy equivalent to FxP with less iterations of the algorithm, proportionally reducing the global energy spent. The 8-bit version of the algorithm becomes more profitable using FlP, which is 80% more accurate with only $1.6\times$ more energy. This paper finally discusses the stake of custom FlP for low-energy general-purpose computation, thanks to its ease of use, supported by an energy overhead lower than what could have been expected.**

I. Introduction

With the exponential improvement of computational abilities and energy efficiency of the last decades, floating-point arithmetic (FlP) has become by far the most used arithmetic, from embedded devices to high-performance computing (HPC). Indeed, the ease of use of FlP arithmetic, its high dynamic range of data and high portability, made it a staple of computer arithmetic. FlP arithmetic is today nearly always used in its normalized 32-bit (float) or 64-bit version (double), natively available in most systems, or even with larger size for very-high accuracy needs. However, using FlP representation for real numbers comes at a cost. Indeed, FlP operations require more area, time and power, especially for large sizes, which potentially implies important shifting. Various IEEE-754 compliant FlP operators were proposed or improved in [1]–[3]. In [4], the authors propose an automated design flow for custom FlP data paths using the FloPoCo (Floating-Point Cores) framework. Given mathematical expressions, the tool generates optimized FlP operators with custom bit-widths and optimized pipelines, mainly targeted for FPGA.

As a consequence, there is a classical association of FlP arithmetic with high-accuracy computing. When high-speed and lower-power computing is sought, fixed-point (FxP) representation tends to be preferred. As a matter of fact, using integer representation for real numbers is less energy and time costly. Moreover, the effects and management of FxP representation to minimize its cost given an accuracy target are well known. In [5], the uniform white noise nature of FxP quantization error is described for the first time. Since then, many publications proposed models and methods to characterize any FxP system error and optimize word-length given an accuracy target. In [6], analytical methods for the evaluation of linear time-invariant systems are proposed. In [7], quantization noise is evaluated for non-recursive non-linear systems. For more complex systems, simulation is used to estimate the output noise. Nevertheless, FxP representation has, by nature, a bad dynamic-accuracy product, as a large dynamic range of manipulated data results in a reduced accuracy for a given word-length.

In this paper, we overcome this segmented approach between FlP and FxP arithmetic, respectively associated to high-accuracy/high-power and low-accuracy/low-power, by studying the impact of using simplified FlP representation with a much lower number of bits than usually, and by dropping safety of IEEE-754 normalization, such as sub-normal consideration and exception handling. For this comparison, a detailed study of the effects of both paradigms is performed, using the K-means clustering algorithm. For this, we use *ApxPerf2.0* [8], a framework based on High-Level Synthesis (HLS) and C++ templates. This framework performs direct comparison of FlP and FxP arithmetic operators for various number representations and also evaluates their impact on performance and accuracy. K-means algorithm has also been a subject for many studies, as a classical fast clustering algorithm. Optimizations of the K-means algorithm for reconfigurable hardware were proposed in [9]–[11]. In these propositions, the parallel iterative nature of K-means clustering algorithm is leveraged to parallelize computations thanks to FPGA resources. As far as we know, there is no study in the literature comparing FlP and FxP for custom size.

This paper is organized as follows. Section II presents the K-means clustering algorithm. FxP and FlP representations are presented in Section III and their advantages and drawbacks are discussed with the help of *ApxPerf2.0*. The accuracy and performance of custom FlP with regards to classical FxP are highlighted. Finally, results on the K-means clustering algorithm using both paradigms are showed and analyzed in Section III. Section V concludes about interests of custom FlP

regarding FxP representation in this case study and closes with what could be expected in a more general purpose.

II. K-Means Clustering Algorithm

This section describes the K-means clustering algorithm. First, the principle of K-means method is described. Then, the specific algorithm used in this case study is detailed.

A. K-Means Clustering Principle

K-means clustering is a well-known method for vector quantization, which is mainly used in data mining, e.g. in image classification or voice identification. It consists in organizing a multidimensional space into a given number of clusters, each being totally defined by its centroid. A given vector in the space belongs to the cluster in which it is nearest from the centroid. The clustering is optimal when the sum of the distances of all points to the centroids of the cluster they belong to is minimal, which corresponds to finding the set of clusters $S = \{S_i\}_{i\in[0,k-1]}$ satisfying

$$\arg\min_{S} \sum_{i=1}^{k} \sum_{x\in S_i} \|x - \mu_i\|^2 , \tag{1}$$

where μ_i is the centroid of cluster S_i. Finding the optimal centroids position of a vector set is mathematically NP-hard. However, iterative algorithms such as Lloyd's algorithm allow us to find good approximations of the optimal centroids by an estimation-maximization process, with a linear complexity (linear with the number of clusters, with the number of data to process, with the number of dimensions and with the number of iterations).

B. K-Means Using Lloyd's Algorithm

The iterative Lloyd's algorithm [12] is used in our case study. It is applied to bidimensional sets of vectors in order to have easier display and interpretation of the results. From now, we will only refer to the bidimensional version of the algorithm. Figure 1 shows results of K-Means on a random set of input vectors, obtained using double-precision FlP computation with a very restrictive stopping condition. This results is considered as the reference golden output in the rest of the paper. The algorithm consists of three main steps:

1) Initialization of the centroids.
2) Data labelling.
3) Centroid position update.

Steps 2 and 3 are iterated until a stopping condition is met. In our case, the main stopping condition is when the difference of the sums of all distances from data points to their cluster's centroid between two iterations is less than a given threshold. A second stopping condition is the maximum number of iterations, required to avoid the algorithm getting stuck when the arithmetic approximations performed are too high to converge. The detailed algorithm for one dimension is given by Algorithm 1. Input data are represented by the vector $data$ of size N_{data}, output centroids by the vector c of size k. The accuracy target for stopping condition is defined by

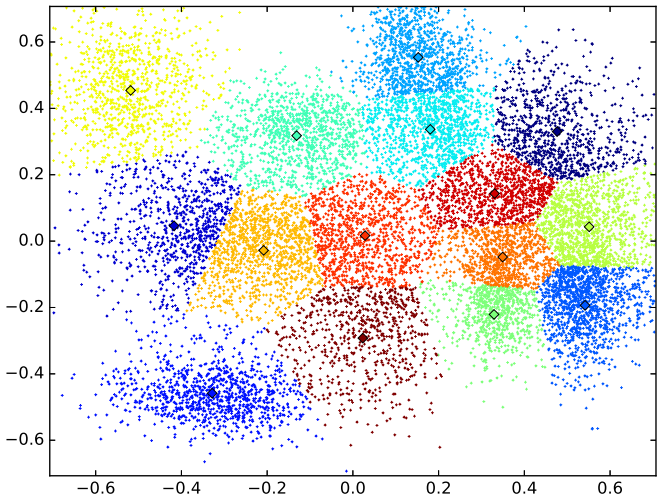


Fig. 1: 2-D K-means clustering golden output example

acc_target and the maximum allowed number of iterations by max_iter. In our study, we use several values for acc_target, and max_iter is set to 150, which is nearly never reached in practice.

The impact of FxP and FlP arithmetic on performance and accuracy is evaluated considering the distance computation function distance_comp, defined by:

$$d \leftarrow (x - y) \times (x - y). \tag{2}$$

Details about accuracy and performance estimation can be found in Section IV.

III. Fixed-Point and Floating-Point Arithmetic

In this study, two paradigms for real number representation are compared: floating-point (FlP) and fixed-point (FxP). Both are often opposed, FlP being the representation the most used in software engineering thanks to its high dynamic range and ease of use. On the other side, FxP evokes simple, fast and energy-efficient computing kernels, which dynamic, accuracy and scaling need to be managed by the system/software designer, costing design time and a certain lack of computing safety (e.g., overflows, underflows). This section compares FlP and FxP in terms of accuracy, performance and hardware cost, and discusses more general advantages and drawbacks.

A. Floating-Point

Thanks to its high dynamic range, ease of use for the programmer and IEEE-754 normalization, most processors now include powerful FlP computing units. This makes FlP representation a high standard for general-purpose computing. A FlP number is represented by three elements: exponent e, mantissa m, and sign bit s, which can also be contained into the mantissa in some representations. The dynamic and accuracy of a FlP representation is intimately linked to the number of bits allocated. The value of a FlP number x_{FlP} is given by:

$$x_{\text{FlP}} = (-1)^s \times m \times 2^e.$$

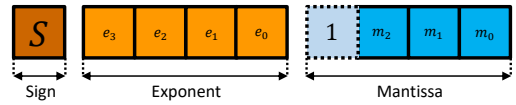
Algorithm 1 K-means clustering (1 dimension)

Require: $k \leq N_{data}$
  $err \leftarrow +\infty$
  $cpt \leftarrow 0$
  $c \leftarrow$ init_centroids$(data)$
  do ▷ Main loop
    $old\_err \leftarrow err$
    $err \leftarrow 0$
    $c\_tmp[0:k-1] \leftarrow 0$
    $min\_distance \leftarrow +\infty$
    for $d \in \{0 : N_{data} - 1\}$ **do**
      $min\_distance \leftarrow +\infty$
      for $i \in \{0 : k-1\}$ **do** ▷ Data labelling
        $distance \leftarrow$ distance_comp$(data[d], c[i])$
        if $distance < min\_distance$ **then**
          $min\_distance \leftarrow distance$
          $labels[d] \leftarrow i$
        end if
      end for
      $c\_tmp[labels[d]] \leftarrow c\_tmp[labels[d]] + data[d]$
      $counts[labels[d]] \leftarrow counts[labels[d]] + 1$
      $err \leftarrow err + min\_distance$
    end for
    for $i \in \{0 : k-1\}$ **do** ▷ Centroids position update
      if $counts[i] \neq 0$ **then**
        $c[i] \leftarrow c\_tmp[i]/counts[i]$
      else
        $c[i] \leftarrow c\_tmp[i]$
      end if
    end for
    $cpt \leftarrow cpt + 1$
  while $(|err - old\_err| > acc\_target) \vee (cpt < max\_iter)$

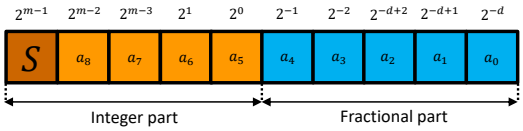
The exponent e can be represented in two's complement or in unsigned with an implicit negative bias. Adding one bit to e not only implies a multiplication by 2 of the dynamic range, but also a division by two of the smallest number. The mantissa m can be represented with an implicit most significant bit (MSB) forced to 1, which saves one bit in the representation. However, it is impossible to represent directly subnormal numbers and zero value. A given value must be dedicated to the representation of zero and a given value of exponent can flag subnormal values. Figure 2a illustrates an 8-bit FlP number with implicit 1 on the mantissa.

Although FlP representation is very convenient and flexible, it is counterbalanced by the cost of arithmetic operations. Indeed, if multiplication has a similar cost to FxP representation, addition has a large overhead since it requires several steps:

- Computation of exponent difference.
- Choice of computation path. *Close path* is activated when the inputs are close enough for their mantissas to recover. Else, the simpler *far path* is activated.
- Mantissa leading zero counting and possible important shifting in the adder close path.



(a) 8-bit floating-point number – $N_e = 4, N_m = 4$



(b) 10-bit fixed-point number – $d = 5$

Fig. 2: Floating-point and fixed-point representations

- Handling of particular cases (subnormals, zero, infinity).
- Result normalization to fit conventions (e.g implicit 1), rounding, exception handling, etc.

The important control overhead of FlP arithmetic can be relaxed by taking some freedom from its strict IEEE 754 version. Therefore, for our study, a custom synthesizable C++ FlP library named *ct_float* was developed, leveraging Mentor Graphics *ac_int*. *ct_float* is embedded in *ApxPerf2.0* framework and open-source. Based on C++ templates, it can be used with any e or m width combination and with different rounding modes. *ct_float* has the following properties:

- implicit 1 for the mantissa,
- no subnormal management for lighter operators,
- zero represented by the nearest from zero negative value,
- no possible over/underflow in the arithmetic operators (saturation to $\{-\infty, -0, +0, +\infty\}$).

ct_float allows most native test or Boolean operations, as well as overloaded addition and multiplication. Addition and multiplication were coded based on classical FlP operations described in [13]. The performance and accuracy of *ct_float* is discussed in Section III-C.

B. Fixed-Point

Fixed-point representation consists in using an integer number to represent a real number. The position of the point to represent this number is implicit and determined at design time. A FxP number x_{FxP} represents a real number x with an integer value x_{int} such as

$$x \simeq x_{\text{FxP}} = x_{\text{int}} \times 2^{-d},$$

where d is the number of bits representing the fractional part of the FxP number. A FxP number on b bits is composed of m bits for the integer part (possibly including the sign bit) and d bits for the fractional part. An example of a 10-bit FxP number is shown in Figure 2b.

Determining the point position for all data in the application is the critical part of fixed-point representation. Indeed, the number of bits representing the integer part must be large enough to accept the data dynamic range to avoid overflows, but must be minimal to leave more bits for the fractional part. The implicit position of the point implies that data alignment and bit dropping must be set at design. In return, arithmetic

operators, especially adder and subtractor, are smaller than for FlP since they are equivalent to integer operators.

C. Floating-Point and Fixed-Point Direct Comparison

Because of the different nature of FlP and FxP errors, this section only compares them in terms of area, delay, and energy. Indeed, FlP error magnitude strongly depends on the amplitude of the represented data. Low-amplitude data have low error magnitude, while high amplitude data have much higher error magnitude. FlP error is only homogeneous considering relative error. Oppositely, FxP has a very homogeneous error magnitude, uniformly distributed with well-known bounds. Thus, its relative error depends on the amplitude of the represented data. It is low for high amplitude data and high for low amplitude data. This duality makes these two paradigms impossible to be atomically compared using the same error metric. The only interesting error comparison which can be performed is applying them on a real-life application, which is done on the K-means clustering algorithm in Section IV.

In all performance studies in this paper, our open-source framework *ApxPerf2.0*, whose flow is described by Figure 3, is used. HLS is achieved by Catapult from Mentor Graphics, RTL synthesis by Synopsys Design Compiler, gate-level simulation by Modelsim leveraging SystemC Verify, and time-based power analysis by Synopsys PrimeTime. A 100 MHz clock is set for designing and estimating performance, and the technology used is 28 nm FDSOI. Energy per operation is

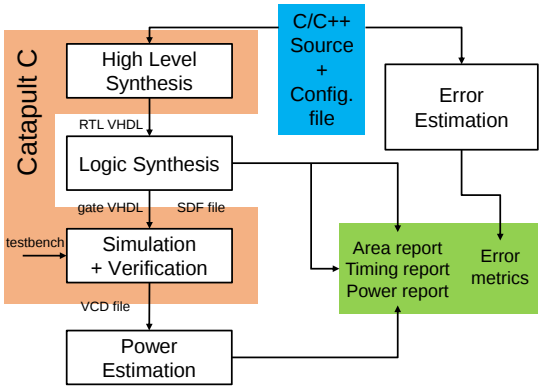


Fig. 3: ApxPerf2.0 framework

estimated using detailed power results given by PrimeTime at gate level. Given the critical path of the design T_c and the clock period T_{clk}, only the energy spent before stabilization is extracted, which allows to have an energy per operation independent of the clock period.

In this section, 8-, 10-, 12-, 14- and 16-bit fixed-width operators are compared. For each of these bit-widths, several versions of the FlP operators are estimated with different exponent widths. 25.10^3 uniform couples of input samples are used for each operator characterization. A tweak ensures that at least 25% of the FlP adder inputs activate the close path of the operator, which has the highest energy by nature. Adders and multipliers are all tested in their fixed-width version, meaning their number of input and output bits are the same. The output is obtained using truncation of the result.

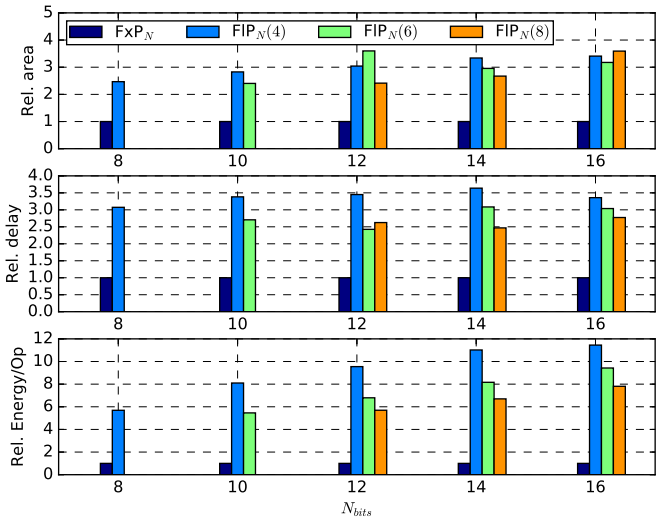


Fig. 4: Relative area, delay and energy per operation comparison between FxP and FlP for different fixed-width adders

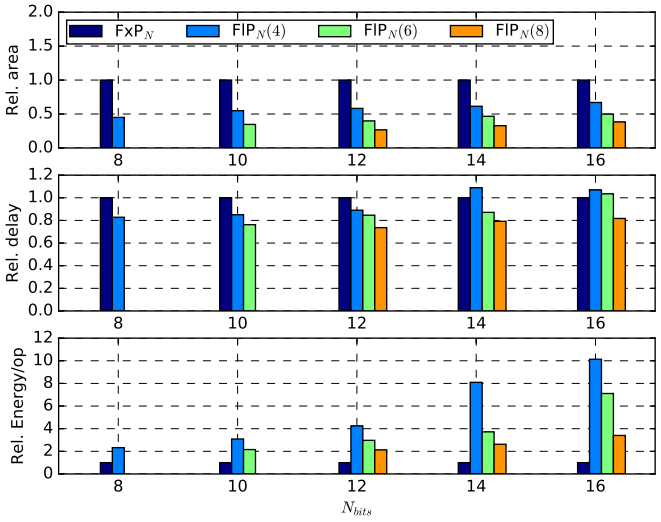


Fig. 5: Relative area, delay and energy per operation comparison between FxP and FlP for different fixed-width multipliers

Figure 4 (resp. Figure 5) shows the area, delay and energy of adders (resp. multipliers) for different bit-widths, relative to the FxP operator. FlP$_N(k)$ represents N-bit FlP with k-bit exponent width. As discussed above, FlP adder has an important overhead compared to FxP adder. For any configuration, results show that area and delay are around $3\times$ higher for FlP. As a consequence, the higher complexity of the FlP adder leads to $5\times$ to $12\times$ more energy per operation.

Results for the multiplier are very different. Indeed, FlP multipliers are 2-3$\times$ smaller than for FxP. Indeed, the control part of FlP multiplier is much less complicated than for the adder. Moreover, as multiplication is applied only on the mantissa, the multiplication is always applied on a smaller number of bits for FlP than for FxP. Timing is also slightly better for FlP, but not as much as area since an important number of operand shifts may be needed during computations.

The impact of these shifts has an important impact on the energy per operation, especially for large mantissas. This brings FlP to suffer an overhead of $2\times$ to $10\times$.

For a good interpretation of these results, it must be kept in mind that in a FxP application, data shifting is often needed at many points in the application. The cost of shifting this data does not appear in the preliminary results presented in this section. However, for FlP, data shifting is directly contained in the operator hardware, which is reflected in the results. Thus, the important advantage of FxP showed by Figures 4-5 must be tempered by the important impact of shifts when applied in applications.

IV. Results on K-Means Clustering

Previous section showed that FxP additions and multiplications consume less energy than FlP for the same bitwidth. However, these results do not yet consider the impact of the arithmetic on accuracy. This section details the impact of accuracy on the bidimensional K-means clustering algorithm presented in Section II-B.

A. Experimental Setup

In the 2D case, the distance computation becomes

$$d \leftarrow (x_0 - y_0) \times (x_0 - y_0) + (x_1 - y_1) \times (x_1 - y_1), \quad (3)$$

which is equivalent to 1 addition, 2 subtractions, and 2 multiplications. However, as distance computation is cumulative on each dimension, the hardware implementation relies only on 1 addition (accumulation), 1 subtraction, and 1 multiplication.

The experimental setup is divided into two parts: accuracy and performance estimation. Accuracy estimation is performed on 20 data sets composed of 15.10^3 bidimensional data samples. These data samples are all generated in a square delimited by the four points $\{\pm\sqrt{2}, \pm\sqrt{2}\}$, using Gaussian distributions with random covariance matrices around 15 random mean points. Several accuracy targets are used to set stopping condition: 10^{-2}, 10^{-3}, 10^{-4}. The reference for accuracy estimation is IEEE-754 floating-point double precision. Figure 1 is an example of a typical golden output for the experiment. The error metrics for the accuracy estimation are: the mean square error of the resulting cluster centroids (CMSE) and the classification error rate (ER) in percents, i.e. the proportion of points not being tagged by the right cluster identifier. The lower the CMSE, the better the estimated position of centroids compared to golden output. Energy estimation is performed using the first of these 20 data sets, limited to 20.10^3 iterations of distance computation for time and memory purposes. As data sets were generated around 15 points, the number of clusters researched is also set to 15. Performance and accuracy of the K-Means clustering experiment, from input data generation to result processing and graphs generation, is fully available in the open-source *ApxPerf2.0* framework, which is used for the whole study.

B. Results

A first study showed that, to get correct results (no artifacts), FlP data must have a minimal exponent width of 5 bits in distance computation (mainly for very small distances) and FxP data a minimal number of 3 bits for its integer part. Thus, all the following results use these parameters. Area, latency and energy of distance computed by Equation 3 are provided. The total energy of the application is defined as

$$E_{\text{K-means}} = E_{\text{dc}} \times (N_{\text{it}} + N_{\text{cycles}} - 1) \times N_{\text{data}}, \quad (4)$$

where E_{dc} is the energy per distance computation calculated from the data extracted with *ApxPerf2.0*, N_{it} the average number of iterations necessary to reach K-means stopping condition, N_{cycles} the number of stages in the pipeline of the distance computation core, and N_{data} the number of processed data per iteration.

Results for 8-bit and 16-bit FlP and FxP arithmetic operators are detailed in Table I, with stopping condition set to 10^{-4}. For

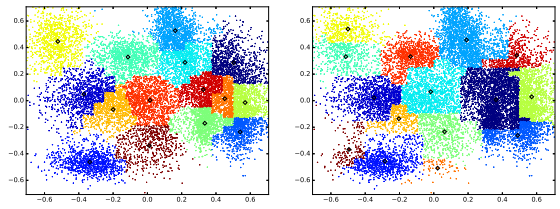
	ct_float$_8(5)$	ct_float$_{16}(5)$	ac_fixed$_8(3)$	ac_fixed$_{16}(3)$
Area (μm^2)	392.3	1148	180.7	575.1
N_{cycles}	3	3	2	2
E_{dc} (nJ)	1.23E−4	5.99E−4	5.03E−5	3.25E−4
N_{it}	8.35	59.3	14.9	65.1
$E_{\text{K-means}}$ (nJ)	38.24	1100	23.90	644.34
CMSE	1.75E−3	3.03E−7	1.85E−2	3.28E−7
Error Rate	35.1 %	2.94 %	62.3 %	0.643 %

TABLE I: 8- and 16-bit performance and accuracy for K-means clustering experiment

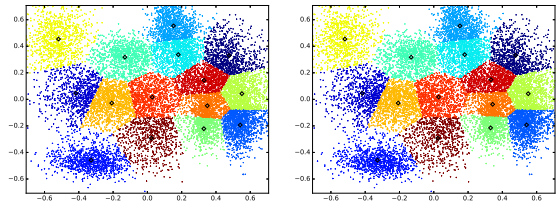
the 8-bit version of the algorithm, several interesting results can be highlighted. First, FlP version is twice as large as FxP version and FlP distance computation consumes $2.44\times$ more energy than FxP. However, FlP version of K-means converges in 8.35 cycles on average against 14.9 for FxP. This makes FlP version for the whole K-means algorithm consuming only $1.6\times$ more energy than FxP. Moreover, FlP version has a huge advantage in terms of accuracy. Indeed, CMSE is $10\times$ better for FlP and ER is $1.8\times$ better. Figures 6a and 6b show the output for FlP and FxP 8-bit computations, applied on the same inputs than the golden output of Figure 1. A very neat stair-effect on data labelling is clearly visible, which is due to the high quantization levels of the 8-bit representation. However, in the FlP version, the positions of clusters centroid is very similar to the reference, which is not the case for FxP.

For the 16-bit version, all results are in favor of FxP, FlP being twice bigger and consuming $1.70\times$ more energy. FxP also provides slightly better error results (2.9% for ER vs. 0.6%). Figures 6c and 6d show output results for 16-bit FlP and FxP. Both are very similar and nearly equivalent to the reference, which reflects the high success rate of clustering.

The competitiveness of FlT over FxP on small bit-widths and the higher efficiency of FxP on larger bit-widths is confirmed by Figure 7 depicting energy vs. classification error rate. Indeed, for different accuracy targets ($10^{-\{2,3,4\}}$), only 8-bit FlP provides higher accuracy for a comparable energy cost, while 10- to 16-bit FxP versions reach an accuracy equivalent



(a) ct_float$_8$(5) (b) ac_fixed$_8$(3)



(c) ct_float$_{16}$(5) (d) ac_fixed$_{16}$(3)

Fig. 6: K-Means clustering outputs for 8- and 16-bit FlP and FxP with accuracy target of 10^{-4}

to FlP with much lower energy. The stopping condition does not seem to have a major impact on the relative performance.

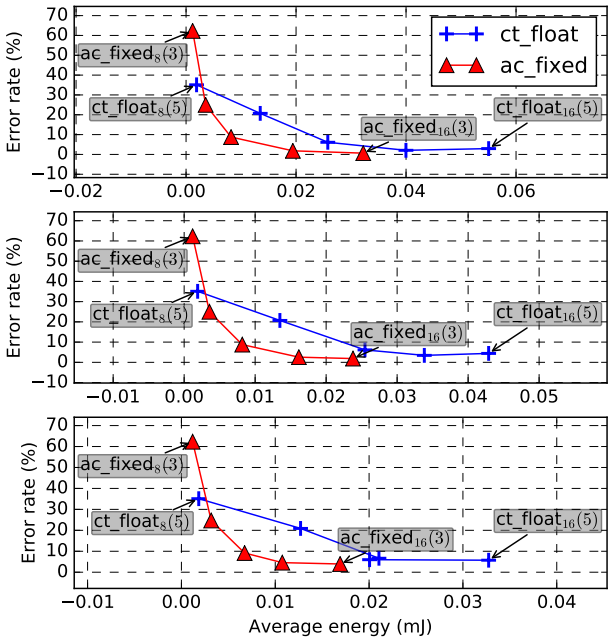


Fig. 7: Energy versus classification error rate for K-means clustering with stopping conditions of 10^{-4} (top), 10^{-3} (center) and 10^{-2} (bottom)

V. Conclusion and Discussion

A raw comparison of FlP and FxP arithmetic operators gives an advantage in area, delay and energy efficiency for FxP. However, the comparison on a real application like the K-means clustering algorithm provides interesting features to custom FlP arithmetic. Indeed, contrary to what would have been expected, FlP arithmetic tends to show better results in terms of energy/accuracy trade-off for very small bit-widths (8 bits in our case). However, increasing this bit-width still leads to an important area, delay and energy overhead of FlP. The most interesting results occur for 8-bit FlP representation. With only 3 bits of mantissa, which corresponds to only 3-bit integer adders and multipliers, the results are better than 8-bit FxP integer operators. This is obviously due to the adaptive dynamic offered by FlP arithmetic at operation level, while FxP has a fixed dynamic which is disadvantageous for low-amplitude data and distance calculation. However, non-iterative algorithms should be tested to know if small FlP keeps its advantage.

From a hardware-design point of view, custom FlP is costly compared to FxP arithmetic. FxP benefits from *free* data shifting between two operators, as outputs of one operator only need to be connected to the inputs of the following in the datapath. However, from a software-design point of view, shifts between FxP computing units must be effectively performed, which leads to a non-negligible delay and energy overhead. Oppositely, FlP computing units do not suffer from this overhead, since data shifting is implemented in the operators and managed by the hardware at runtime. Thanks to this feature, FlP exhibits another important advantage which is the ease of use, since software development is faster and more secured.

Hence, in the aim of producing general-purpose low-energy processors, small-bitwidth FlP arithmetic can provide major advantages compared to classical integer operators embedded in microcontrollers, with a better compromise between ease of programming, energy efficiency and computing accuracy.

References

[1] A. Beaumont-Smith, N. Burgess, S. Lefrere, and C. C. Lim, "Reduced latency ieee floating-point standard adder architectures," in *Proc. of 14th IEEE Symposium on Computer Arithmetic*, 1999, pp. 35–42.

[2] J. Thompson, N. Karra, and M. J. Schulte, "A 64-bit decimal floating-point adder," in *IEEE Computer Society Annual Symposium on VLSI*, Feb 2004, pp. 297–298.

[3] A. M. Nielsen, D. W. Matula *et al.*, "An ieee compliant floating-point adder that conforms with the pipeline packet-forwarding paradigm," *IEEE Trans. on Computers*, vol. 49, no. 1, pp. 33–47, Jan 2000.

[4] F. de Dinechin and B. Pasca, "Designing custom arithmetic data paths with flopoco," *IEEE Design Test of Computers*, vol. 28, no. 4, pp. 18–27, July 2011.

[5] B. Widrow, I. Kollar, and M.-C. Liu, "Statistical theory of quantization," *IEEE Trans. on Instrumentation and Measurement*, vol. 45, no. 2, pp. 353–361, April 1996.

[6] D. Menard, R. Rocher, and O. Sentieys, "Analytical Fixed-Point Accuracy Evaluation in Linear Time-Invariant Systems," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 55, no. 1, Nov. 2008.

[7] D. Menard, R. Rocher *et al.*, "Automatic SQNR determination in non-linear and non-recursive fixed-point systems," in *12th European Signal Processing Conference*, Sept 2004, pp. 1349–1352.

[8] B. Barrois, O. Sentieys, and D. Menard, "The hidden cost of functional approximation against careful data sizing – a case study," in *Design, Automation Test in Europe (DATE)*, March 2017, pp. 181–186.

[9] W.-C. Liu, J.-L. Huang, and M.-S. Chen, "Kacu: k-means with hardware centroid-updating," in *Conf. on Emerging Information Technology 2005.*, Aug 2005.

[10] H. M. Hussain, K. Benkrid *et al.*, "Fpga implementation of k-means algorithm for bioinformatics application: An accelerated approach to clustering microarray data," in *2011 NASA/ESA Conf. on Adaptive Hardware and Systems (AHS)*, June 2011, pp. 248–255.

[11] J. S. S. Kutty, F. Boussaid, and A. Amira, "A high speed configurable fpga architecture for k-mean clustering," in *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2013, pp. 1801–1804.

[12] S. Lloyd, "Least squares quantization in pcm," *IEEE Trans. on Information Theory*, vol. 28, no. 2, pp. 129–137, March 1982.

[13] J.-M. Muller, N. Brisebarre, F. De Dinechin *et al.*, *Handbook of floating-point arithmetic*. Springer Science & Business Media, 2009.