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

Benjamin Barrois and Olivier Sentieys

IRISA/INRIA – Cairn team
University of Rennes olivier.sentieys@irisa.fr

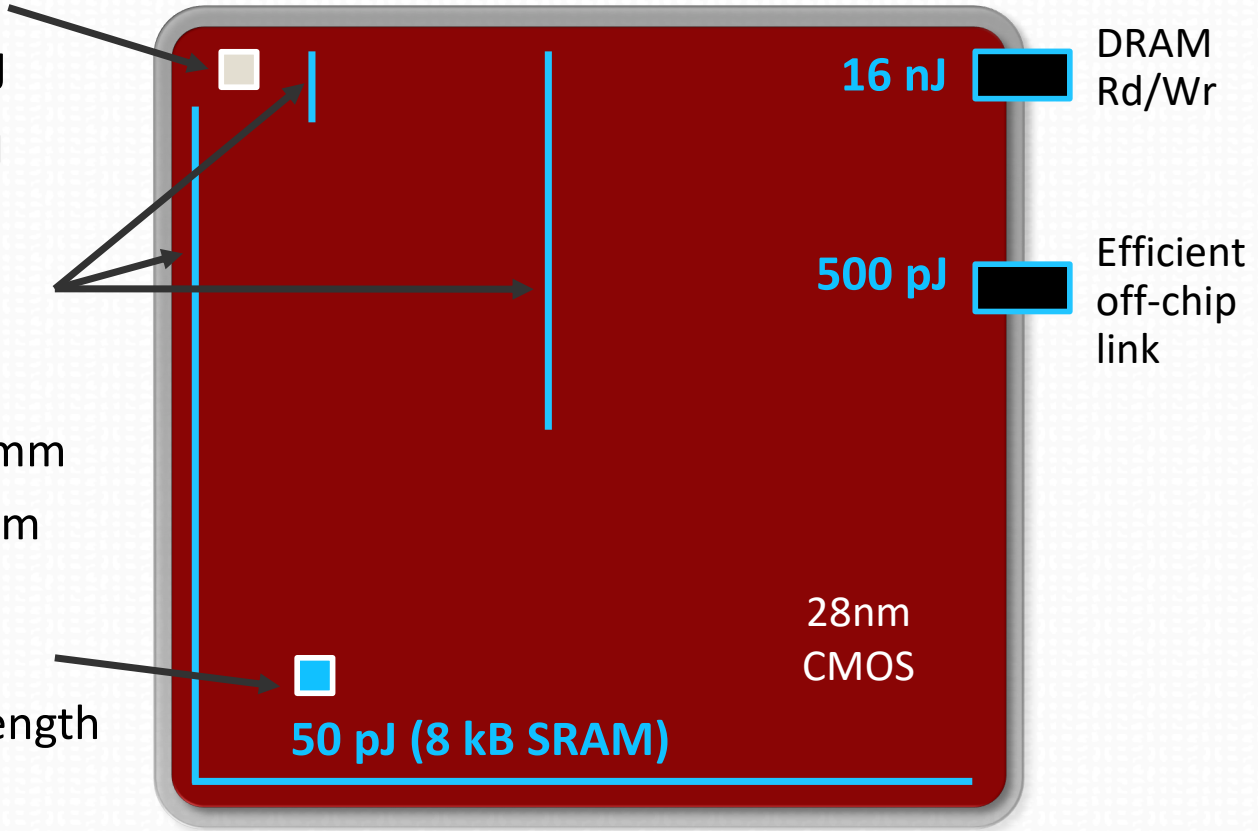
Energy Cost in a Processor/SoC

- 64-bit FPU: 20pJ/op
- 32-bit addition: 0.05pJ
- 16-bit multiply: 0.25pJ

- Wire energy
  - 240fJ/bit/mm per ⇵
  - 32 bits: 40pJ/word/mm
  - 8 bits: 10pJ/word/mm

- Memory/Register-File
  - Depends on word-length

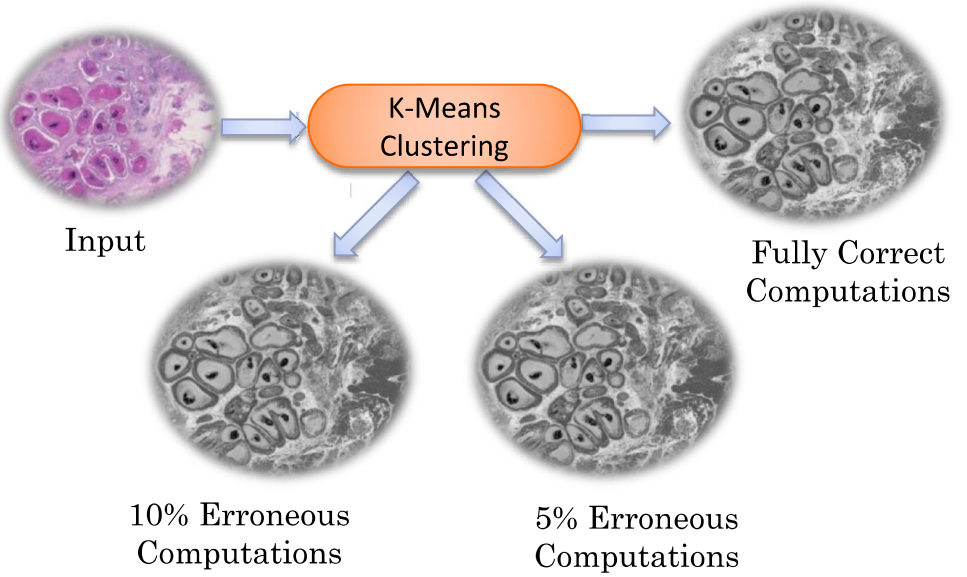
[Adapted from Dally, IPDPS'11]



Energy strongly depends on data representation and size

Many Applications are Error Resilient

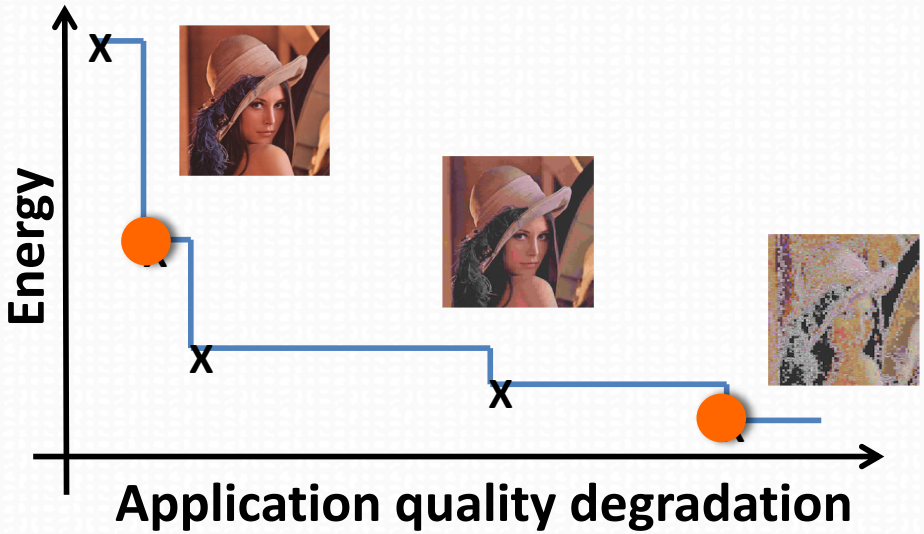
- Produce outputs of acceptable quality despite approximate computation
  - Perceptual limitations
  - Redundancy in data and/or computations
  - Noisy inputs
- Digital communications, media processing, data mining, machine learning, web search, …



e.g. Image Segmentation

Approximate Computing

- Play with number representation to reduce energy and increase execution speed while keeping accuracy in acceptable limits
  - Relaxing the need for fully precise operations
- Trade quality against performance/energy
  - Compile-time/run-time
- Different levels
  - Operators/functions/algorithms



Outline

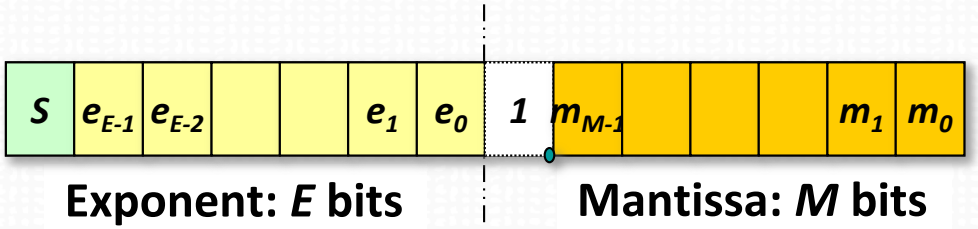
- Introduction
- Number Representation
  - Fixed-Point
  - Floating-Point
- Customizing Arithmetic Operators
- Direct Comparison of Custom Operators
- ApxPerf Framework
- Results on K-Means Clustering Algorithm
- Conclusions

Number Representation

- Floating-Point (FlP)

$$x = (-1)^s \times m \times 2^{e-127}$$

s: sign, *m*: mantissa, *e*: exponent



  - Easy to use
  - High dynamic range
  - IEEE 754

Format	e	m	bias
Single Precision	8	23	127
Double Precision	11	52	1023

- Fixed-Point (FxP)

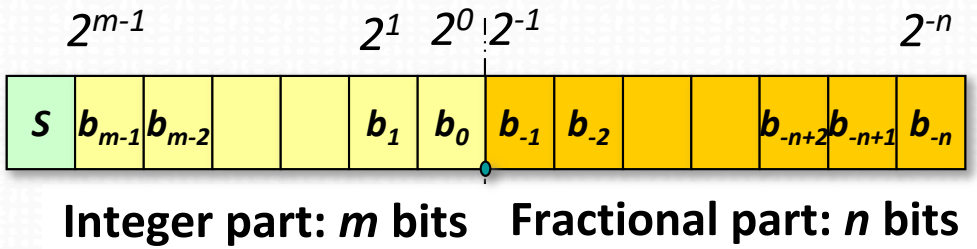
$$x = p \times K$$

p: integer, $K=2^{-n}$: fixed scale factor

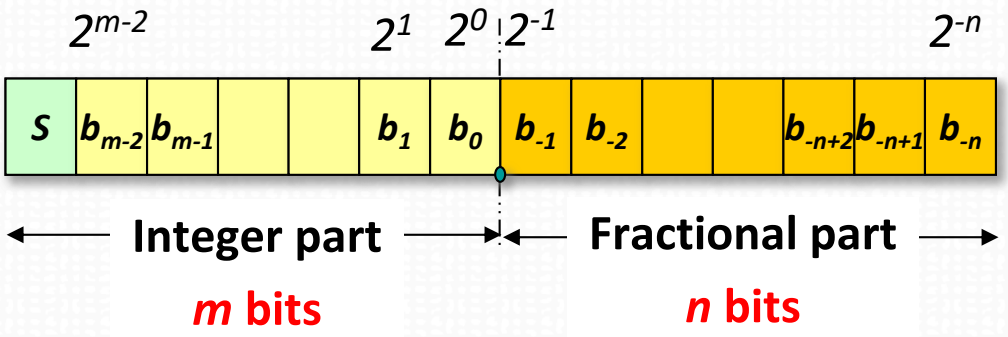
  - Integer arithmetic
  - Efficient operators
    - Speed, power, cost

$$x = s.(-2)^m + \sum_{i=-n}^{m-1} b_i.2^i$$

s: sign, m: magnitude, n: fractional



Fixed-Point Arithmetic



$$x = s.(-2)^{m-1} + \sum_{i=-n}^{m-2} b_i.2^i$$

s: sign, m: magnitude, n: fractional

- Accuracy (error)

```
   1.640625
 * 2.3125
 = 3.7939453125
```

- Dynamic range

```
   1.640625
 + 2.5125
 = 4.153125
```

$x \in [-4; 4[$

overflow

- Need for explicit normalization

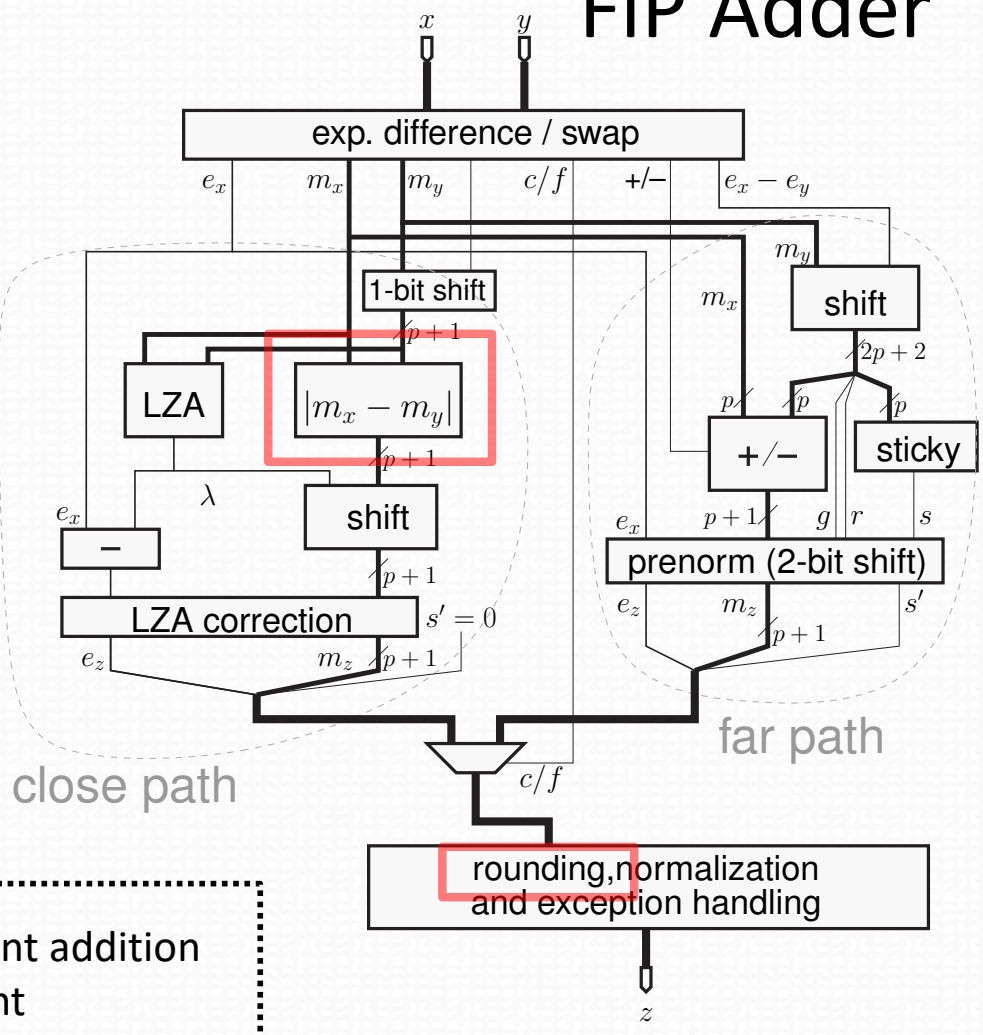
```
(int)(((INT64)a * (INT64)b) >> N)
```

- Use of popular libraries (e.g. sc_fixed, ac_fixed)

Floating-Point Arithmetic

- Floating-point hardware is doing the job for you!
- Arithmetic operators are therefore more complex

FlP Adder

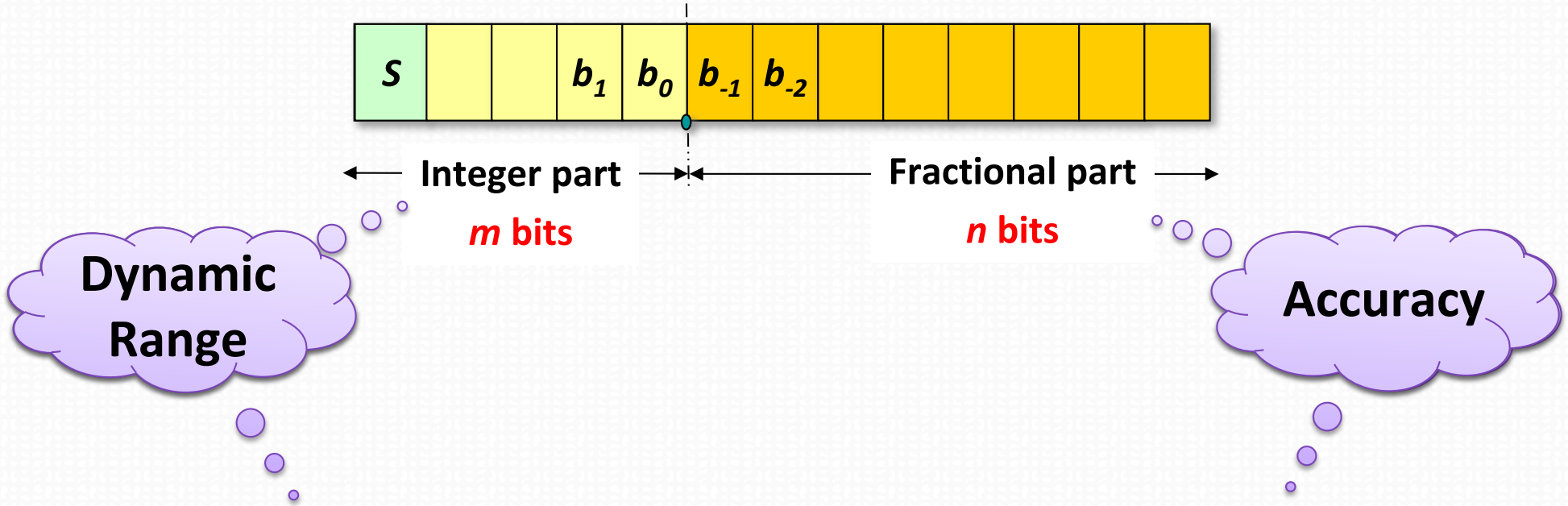


Fixed-point addition equivalent

J.-M. Muller et al., Handbook of Floating-point arithmetic, Springer, 2009.

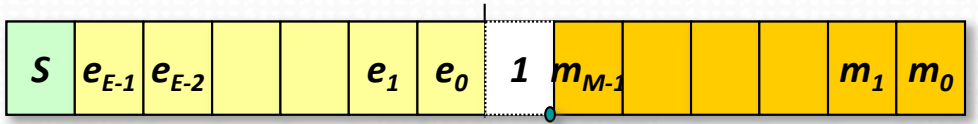
Customizing Fixed-Point

- Minimize word-length *W=m+n*
- Determine integer and fractional parts
  - Fixed-point refinement



Customizing Floating-Point

- Minimize word-length *W=E+M+1*
- Determine exponent and mantissa (and bias)
- Error is relative to number value



Exponent

***E* bits**

Mantissa

***M* bits**

Range & Accuracy

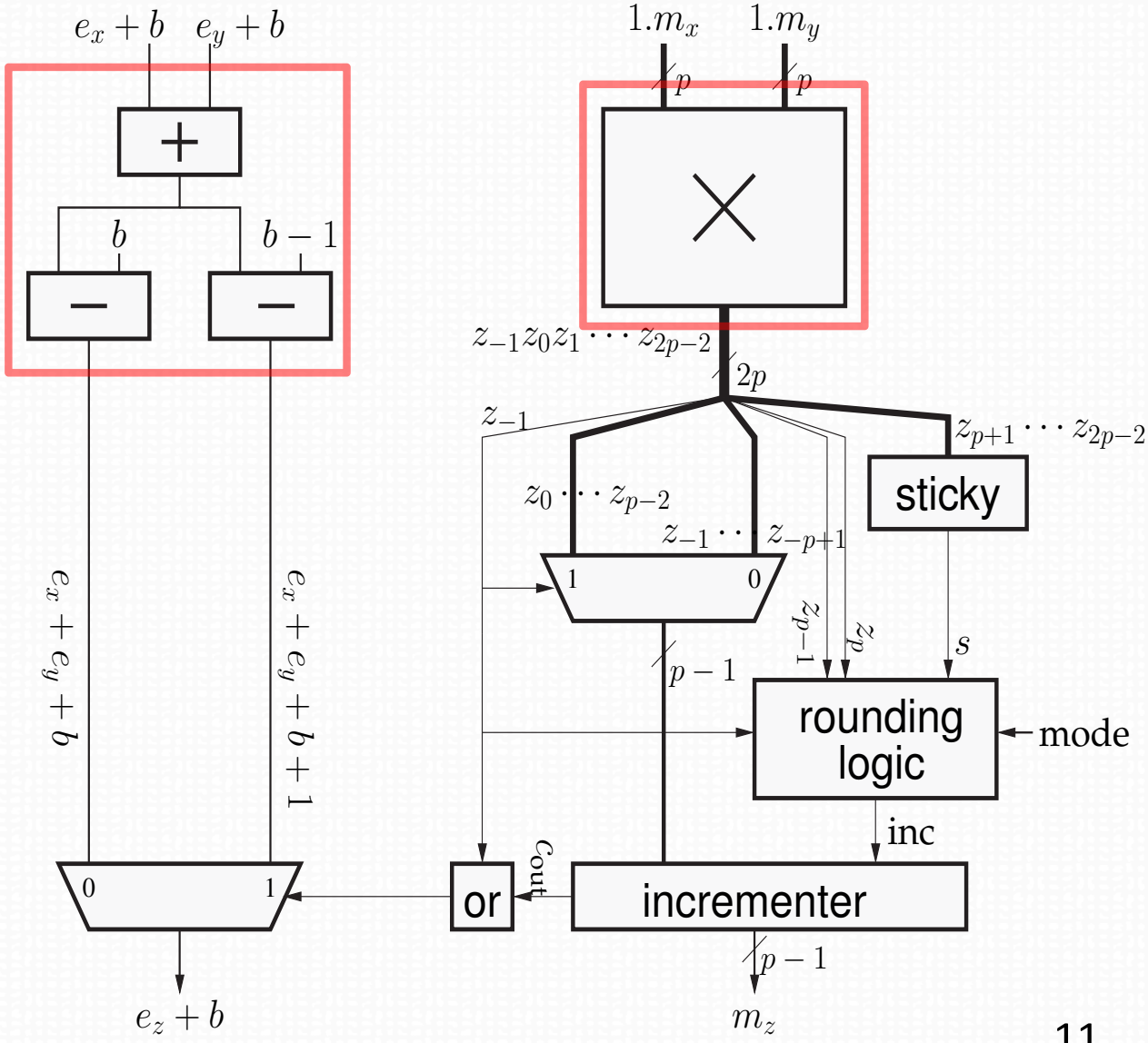
Ensure no overflow and accuracy of small numbers

Accuracy

Provide a minimal numerical accuracy

Floating-Point Multiplication

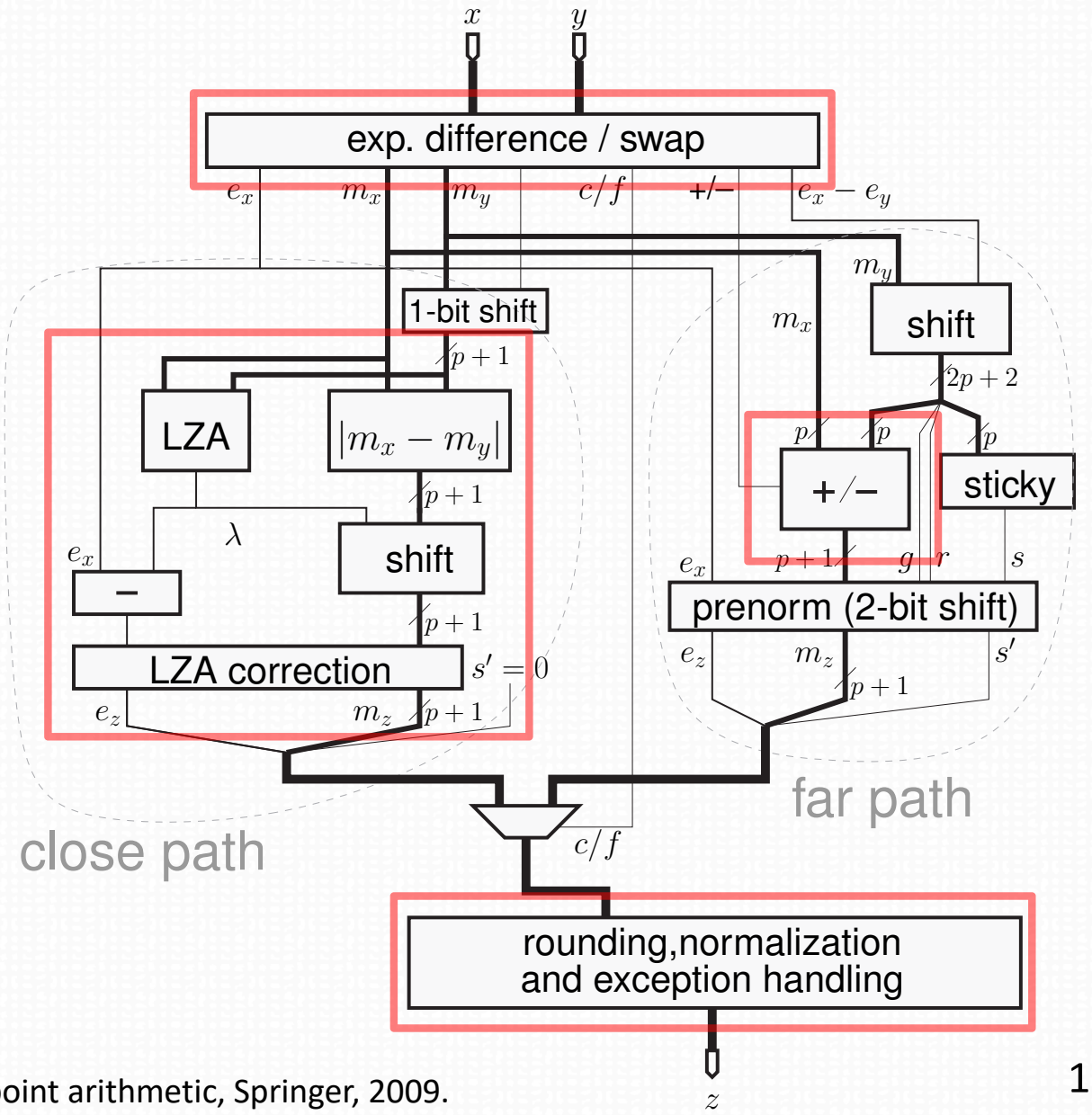
- Representation *(W,E,M)*
- Exponent *e*
  - *E* bits
- Mantissa *m*
  - *M* bits



J.-M. Muller et al., Handbook of Floating-point arithmetic, Springer, 2009.

Floating-Point Addition

- Representation *(W,E,M)*
- Exponent *e*
  - *E* bits
- Mantissa *m*
  - *M* bits



J.-M. Muller et al., Handbook of Floating-point arithmetic, Springer, 2009.

ct_float: a Custom-FlP C++ Library

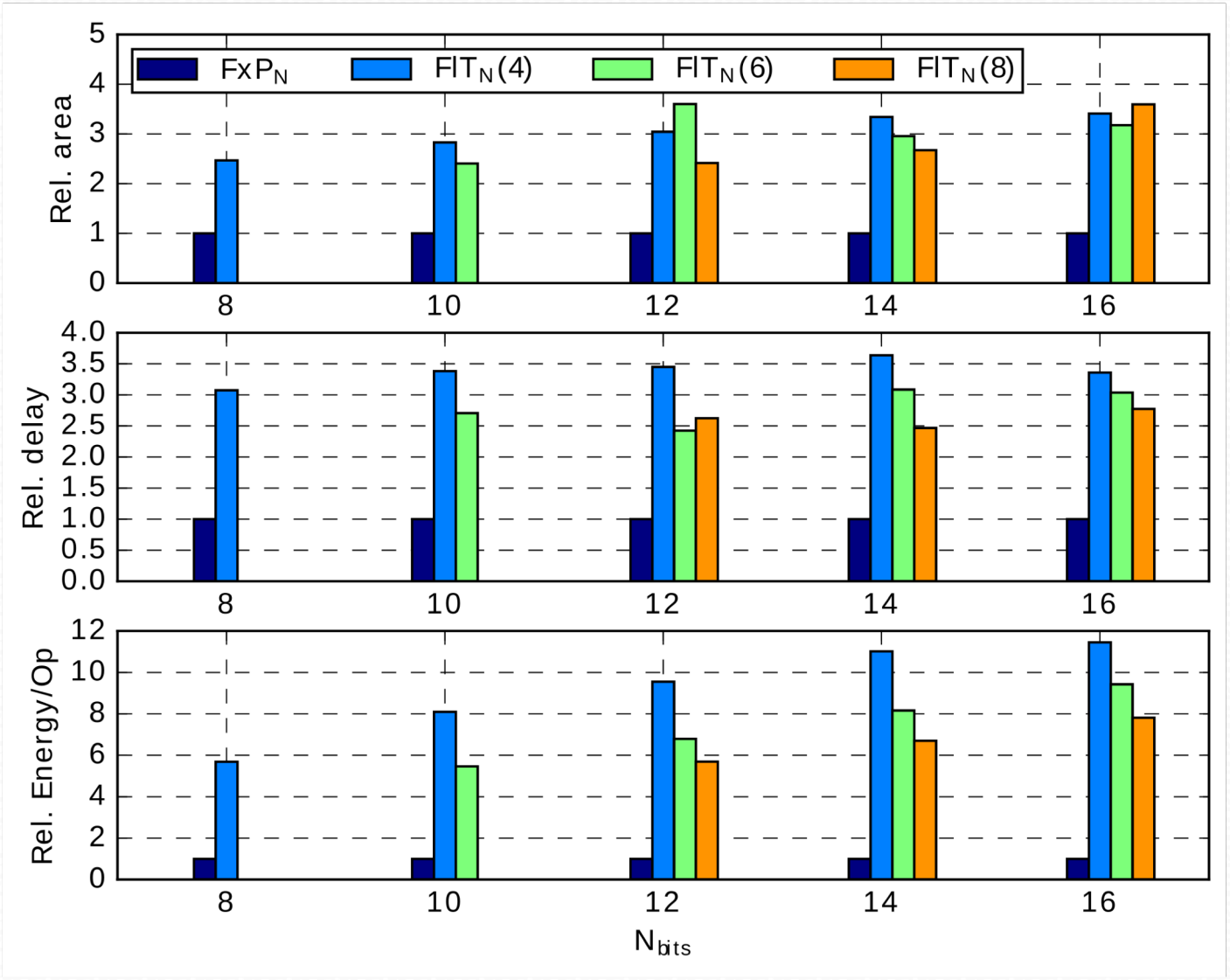
- ct_float: a Custom Floating-Point C++ Library
  - Operator simulation and (High-Level) synthesis
  - Templated C++ class
    - Exponent width e (int)
    - Mantissa width m (int)
    - Rounding method r (CT_RD,CT_RU,CT_RND,CT_RNU)
  - Many synthetizable overloaded operators
    - Comparison, arithmetic, shifting, etc.

```
ct_float<8,12,CT_RD> x,y,z;
x = 1.5565e-2;
z = x + y;
```

FxP vs. FlP: Adders

- FxP_N
  - Fixed-Point
  - *N* bits
- $FlT_N(E)$
  - Floating-Point
  - *N* bits
  - Exponent *E* bits

- FxP adders are always smaller, faster, less energy

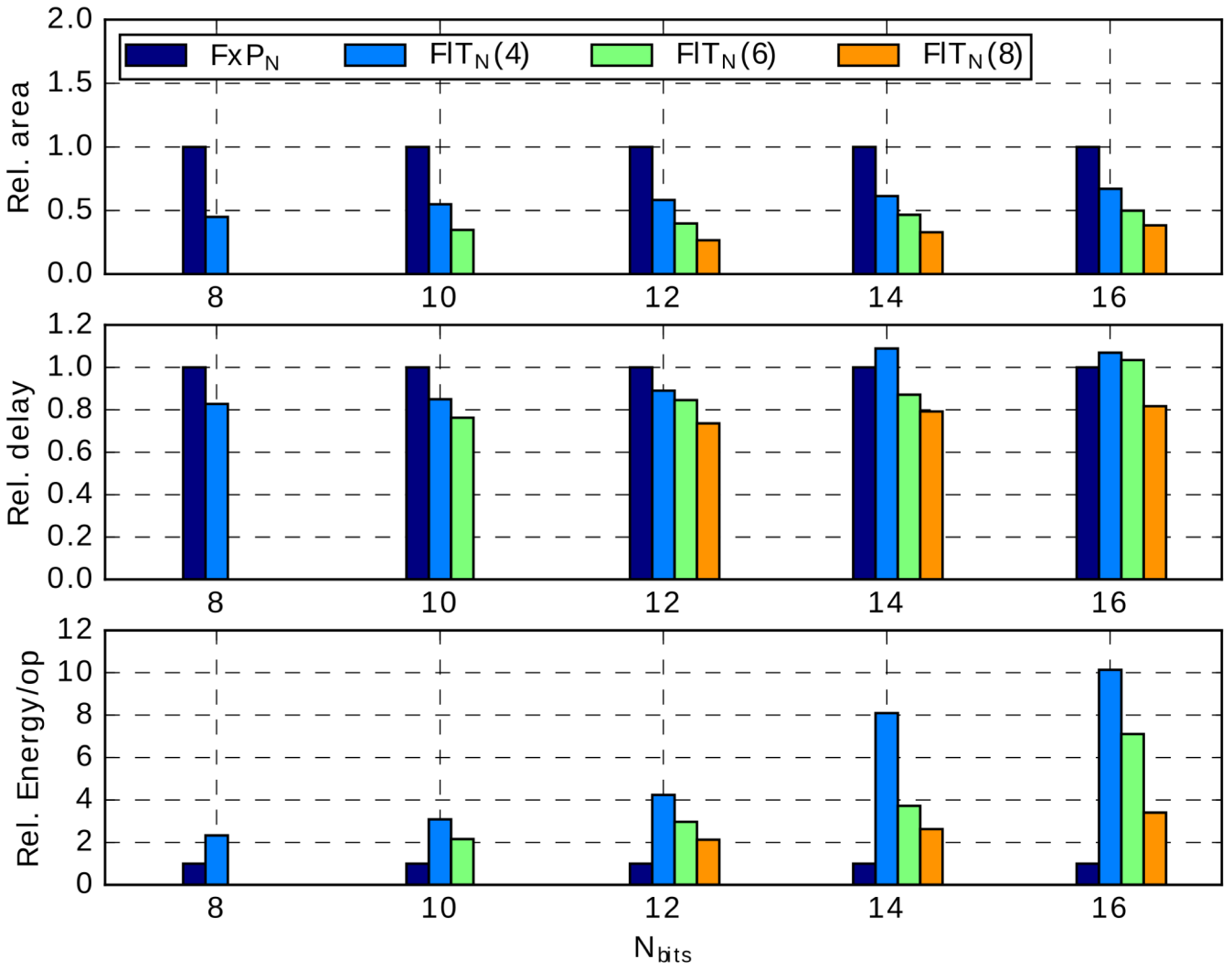


28nm FDSOI technology, Catapult (HLS), Design Compiler, PrimeTime

FxP vs. FlP: Multipliers

- FxP_N
  - Fixed-Point
  - *N* bits
- $FlT_N(E)$
  - Floating-Point
  - *N* bits
  - Exponent *E* bits

- FlP multipliers are smaller, faster, but more consuming



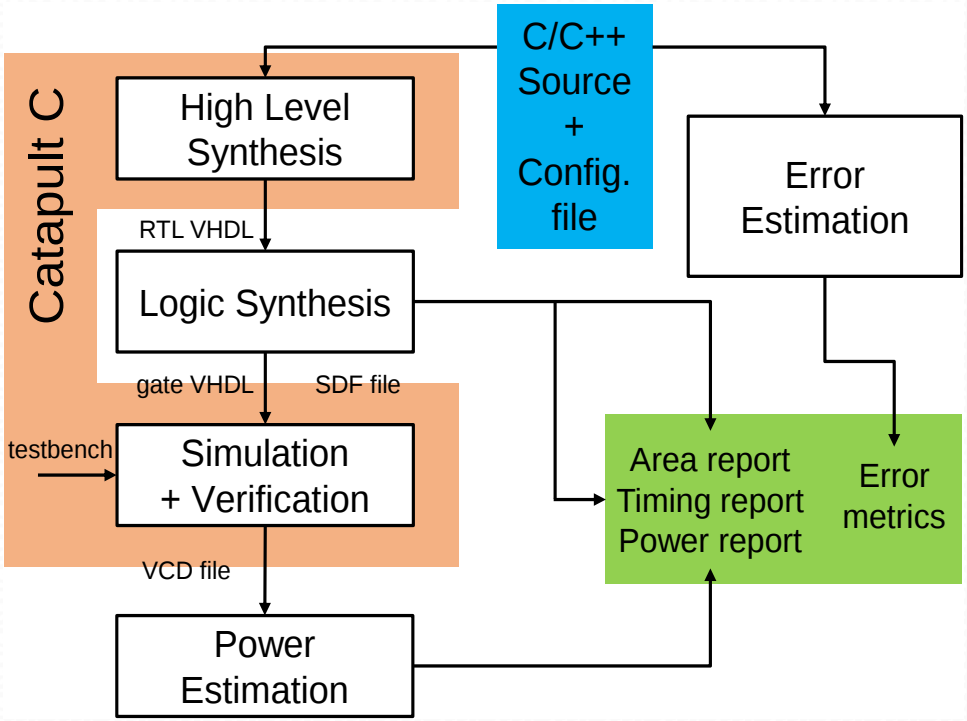
28nm FDSOI technology, Catapult (HLS), Design Compiler, PrimeTime

Outline

- Introduction
- Number Representation
  - Fixed-Point
  - Floating-Point
- Customizing Arithmetic Operators
- Direct Comparison of Custom Operators
- ApxPerf Framework
- Results on K-Means Clustering Algorithm
- Conclusions

Energy-Accuracy Trade-offs

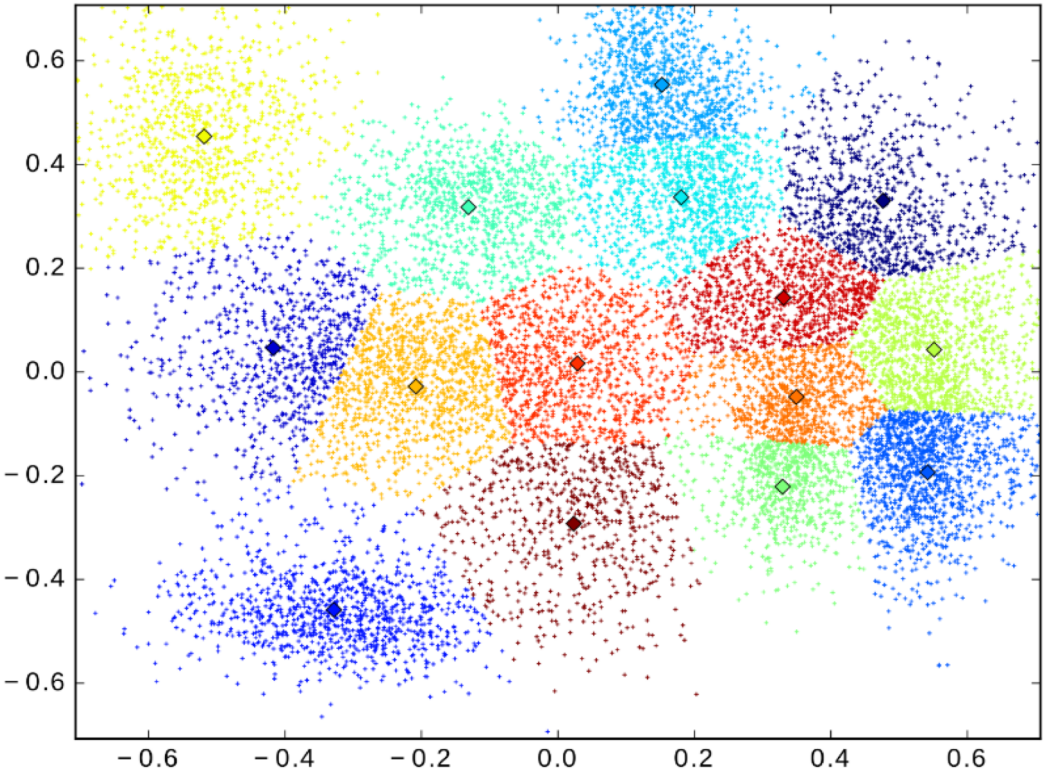
- ApxPerf2.0 framework
  - Based on C++ templates, HLS, and Python
  - VHDL and C/C++ operator descriptions
    - Approximate, FxP, FlP



  - Fully automated characterization
  - Generates delay, area, and power results
  - Extract error metrics
    - mean square error, mean average error, relative error, min/max error, bit error rate, etc.

K-Means Clustering

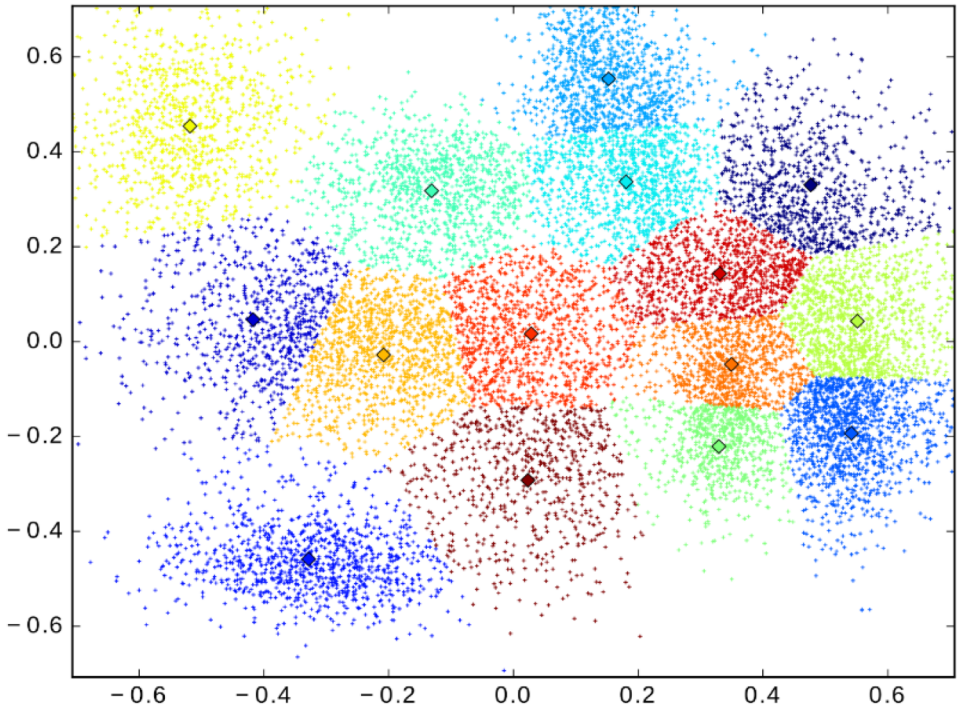
- Data mining, image classification, etc.
- A multidimensional space is organized as:
  - k clusters S_i,
  - S_i defined by its centroid μ_i



- Finding the set of clusters $S = \{S_i\}_{i \in [0,k-1]}$ satisfying $\arg\min_S \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|^2$ is NP-hard

K-Means Clustering

- Lloyd's iterative algorithm
  - approximations of the optimal centroids
  - estimation-maximization three-step iterative process



- Distance computation $d \leftarrow (x - y) \times (x - y)$
- Iteration of computations until
  - sum of distances from data points *x* to centroid μ_i between two iterations is less than a given threshold
  - maximum number of iterations

Approximate K-Means Clustering

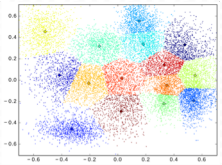
- Experimental setup
  - 20 data sets composed of 15.10^3 samples
  - Gaussian distributions with random covariance matrices around 15 random mean points
  - Accuracy targets: 10^{-2}, 10^{-3}, 10^{-4}
  - Reference is double-precision floating-point
  - 28nm, 100MHz
- Error metrics
  - Mean square error of cluster centroids (CMSE)
    - lower is better
  - Classification error rate (ER)
    - i.e. proportion of points not being tagged by cluster identifier

Approximate K-Means Clustering

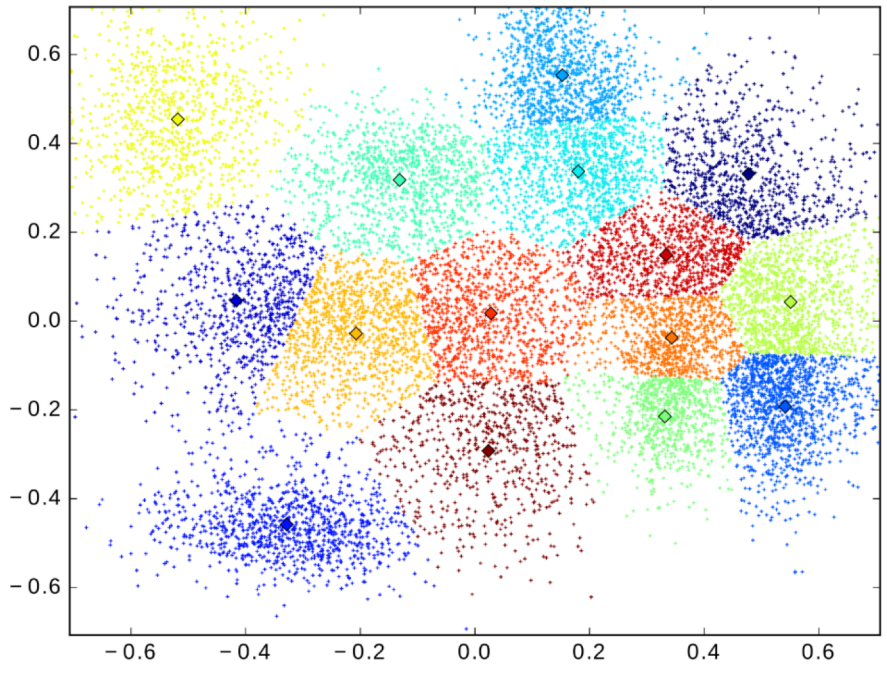
- Results with 8-bit and 16-bit FlP and FxP arithmetic operators
- Stopping condition set to 10^{-4}

	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 %

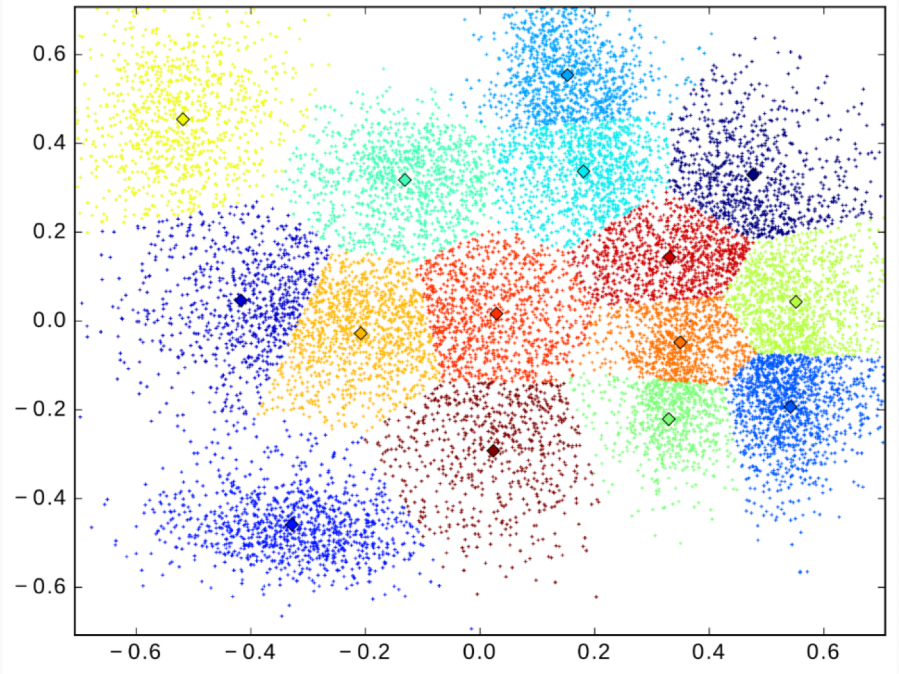
Approximate K-Means Clustering



- W = 16 bits, accuracy = 10^{-4}
- No major (visible) difference with reference

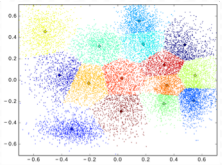


ac_fixed_{16}
3-bit integer part
13-bit fractional part

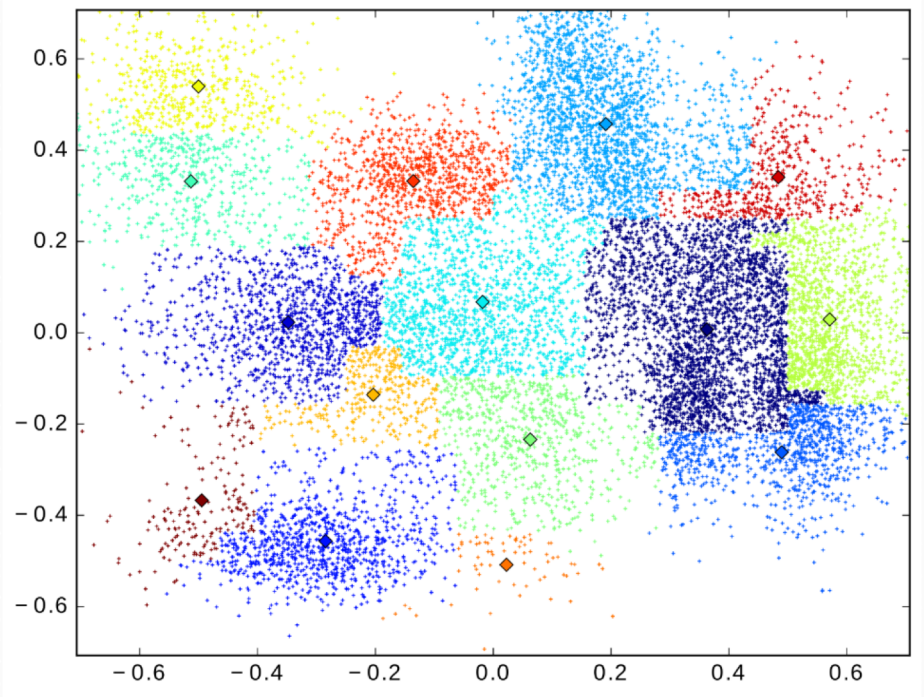


ct_float_{16}
5-bit exponent
11-bit mantissa

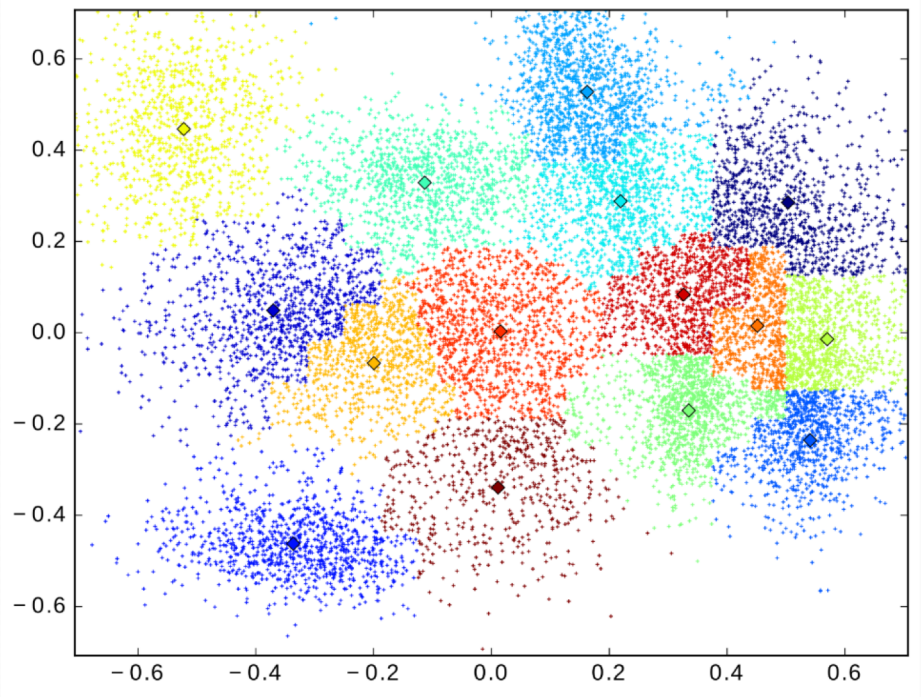
Approximate K-Means Clustering



- W = 8 bits, accuracy = 10^{-4}
- 8-bit float is better and still practical



ac_fixed_8
3-bit integer part
5-bit fractional part

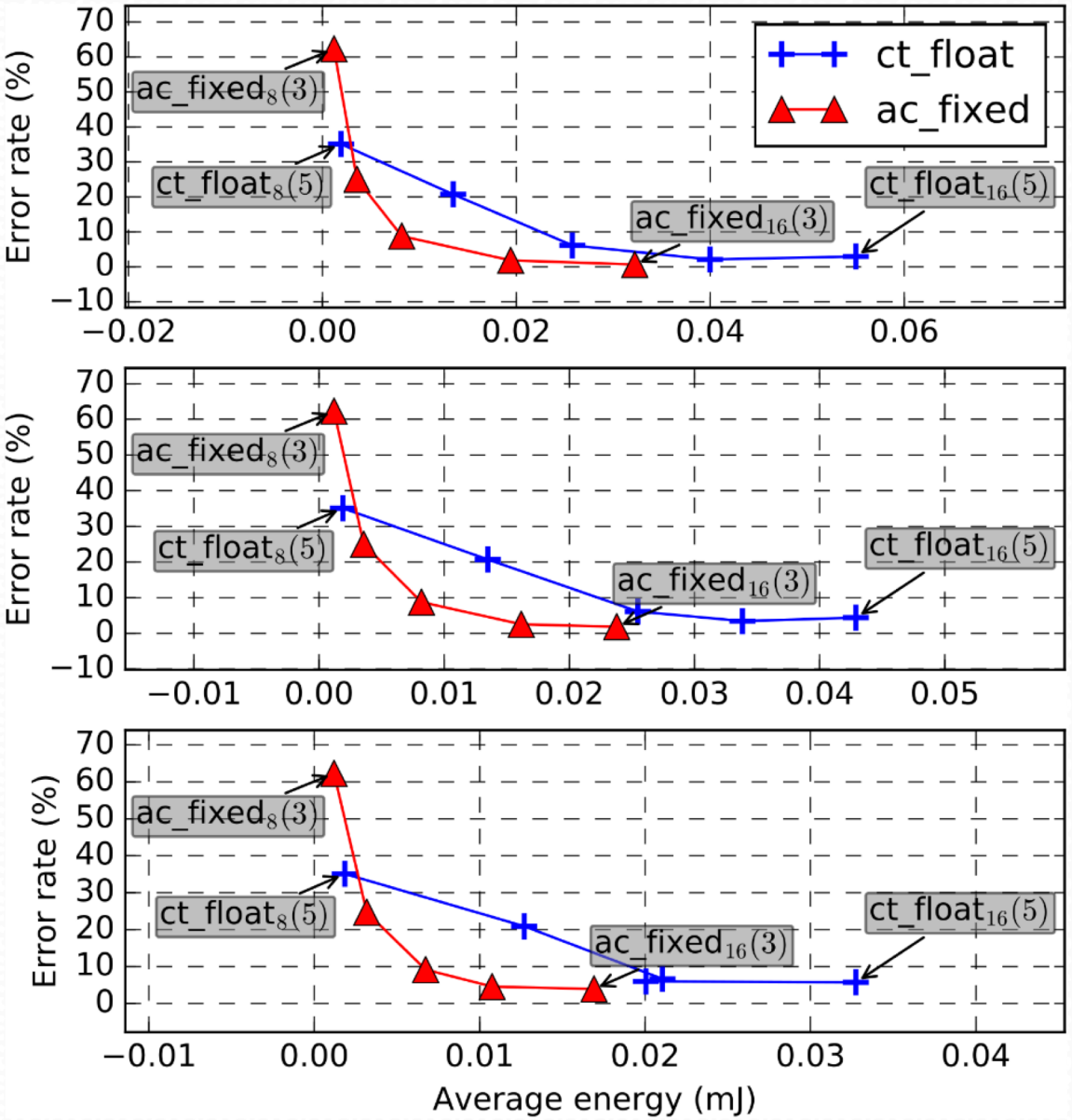


ct_float_8
5-bit exponent
3-bit mantissa

Energy versus Classification Error Rate

- Average energy consumed by K-means algorithm
- Stopping conditions:

  10^{-2} to 10^{-4}

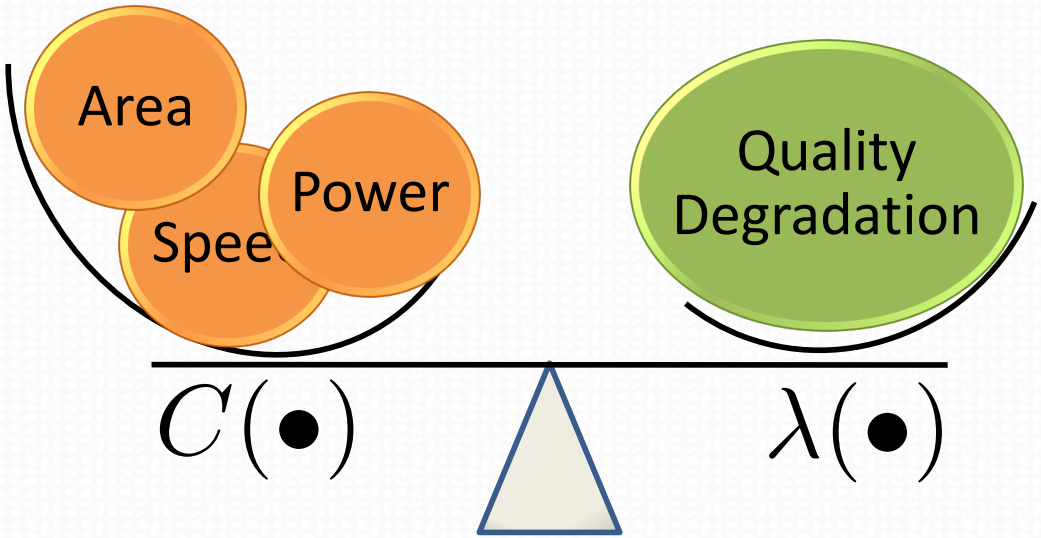


Conclusions

- Total energy (algorithm) depends on:
  - Energy of arithmetic operations
  - Algorithm convergence speed
- Slower increase of errors for floating-point
- Small floating-point (e.g. 8-bit) provides better error rate/energy ratio
- Perspectives
  - Custom exponent bias in *ct_float*
  - Towards an automatic optimizing compiler considering both FxP and FlP representations

Customizing Number Representation

- Loss of accuracy incurs quality degradation
- Essentially, an optimization process
  - Determine the number of bits for each data
  - Determine the format for each data



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

Benjamin Barrois and Olivier Sentieys

IRISA/INRIA – Cairn team
University of Rennes olivier.sentieys@irisa.fr