Playing with number representations and operator-level approximations

Olivier Sentieys

Playing with number representations and operator-level approximations

Olivier Sentieys

INRIA

Univ Rennes

olivier.sentieys@inria.fr

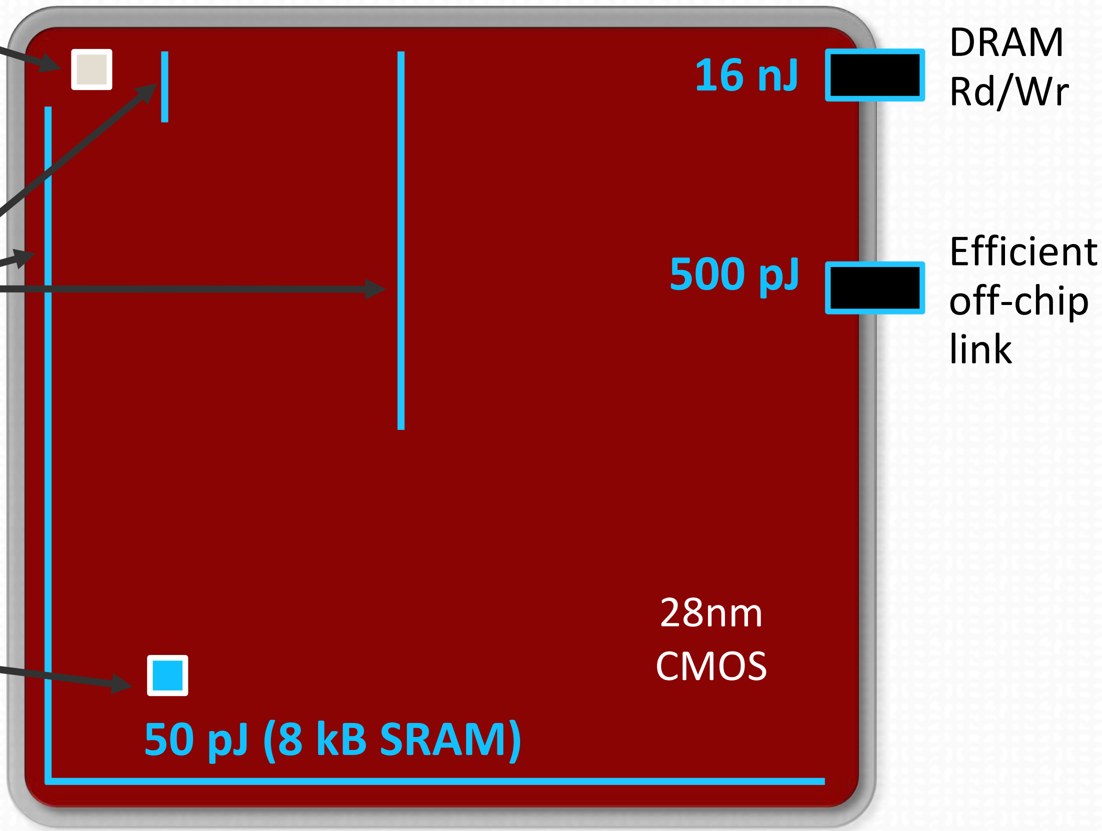
Energy Cost in a Processor/SoC

- 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]

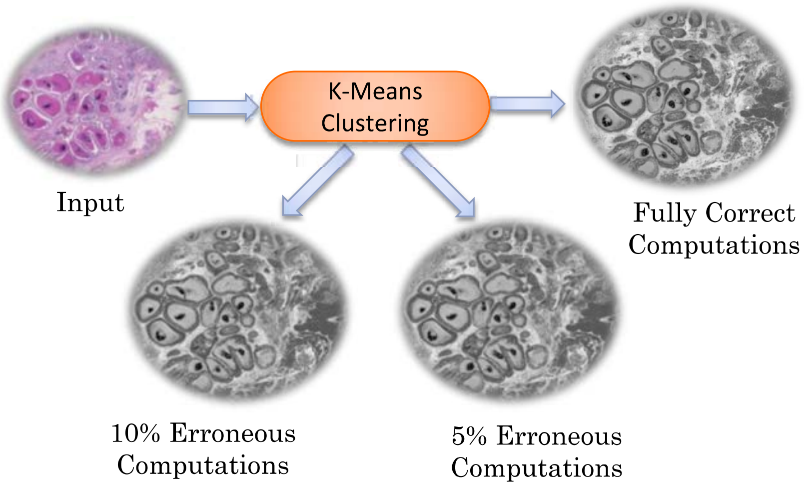


Energy strongly depends on data representation and size

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, …

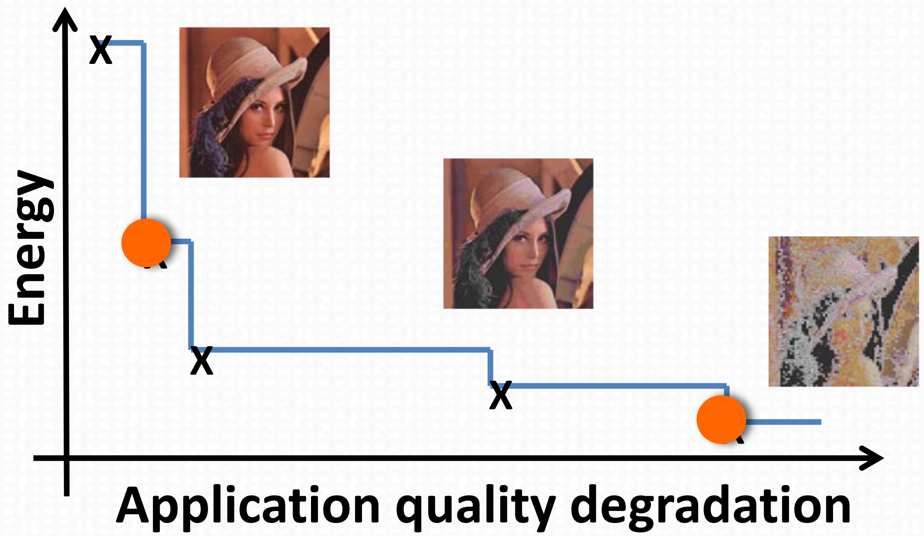


e.g. Image Segmentation

Approximate Computing

- Play with number representations 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
  - Design-time/run-time
- Different levels
  - Operators/functions/algorithms



Outline

- Motivations for approximate computing
- Number representations
- Approximate operators or careful rounding?
- Operator-level support for approximate computing
- Stochastic computing
- Conclusions

Outline

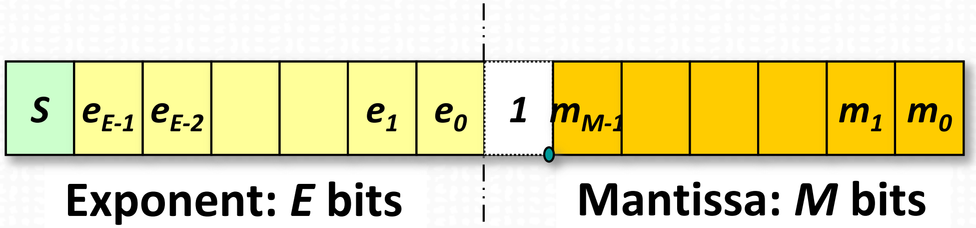
- Motivations for approximate computing
- Number representations
  - Fixed-Point
  - Floating-Point
  - Customizing Arithmetic Operators
  - ApxPerf Framework
- Approximate operators or careful rounding?
- Operator-level support for approximate computing
- Stochastic computing
- Conclusions

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

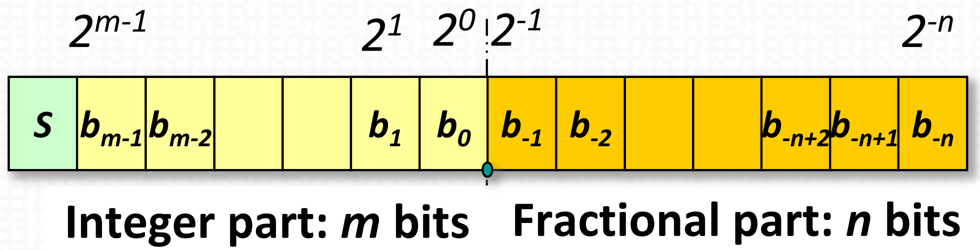
Format	e	m	bias
Single Precision	8	23	127
Double Precision	11	52	1023

- Fixed-Point (FxP)

$$x = p \times K$$

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

  - Integer arithmetic
  - Efficient operators
    - Speed, power, cost
  - Hard to use...

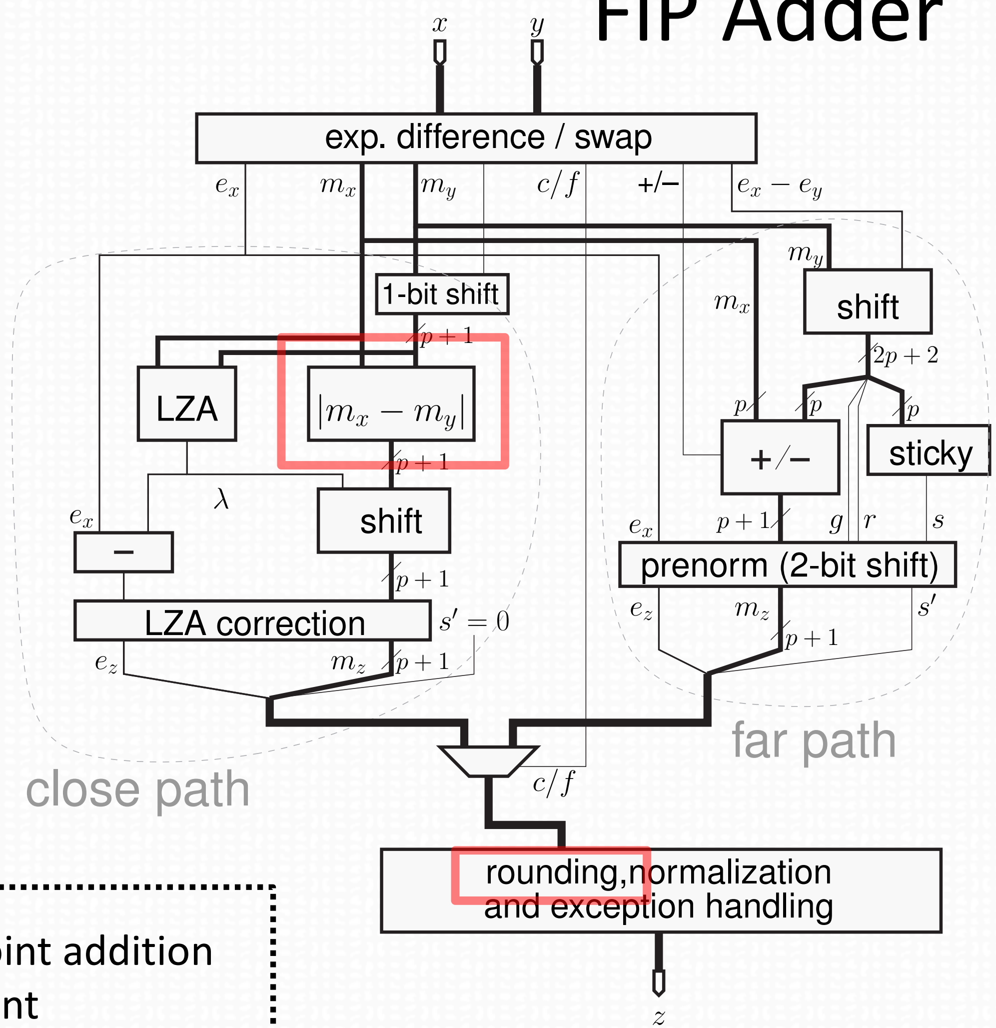


Floating-Point Arithmetic

- Floating-point hardware is doing the job for you!

- FlP operators are therefore more complex

FlP Adder

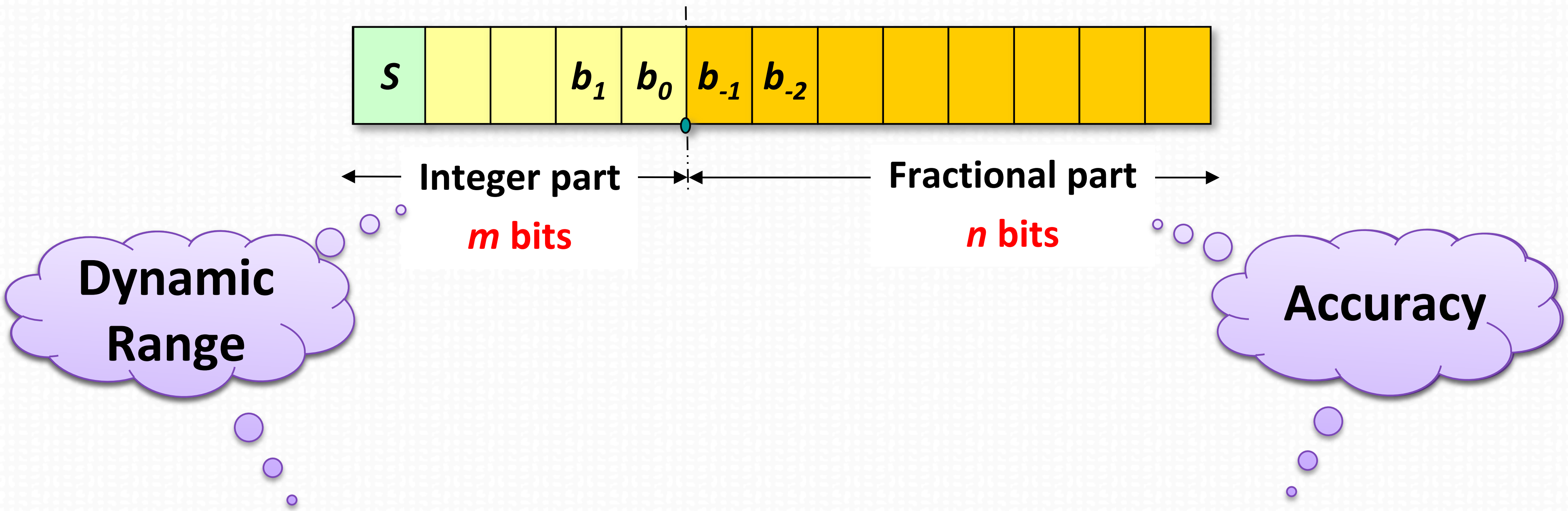


Fixed-point addition equivalent

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

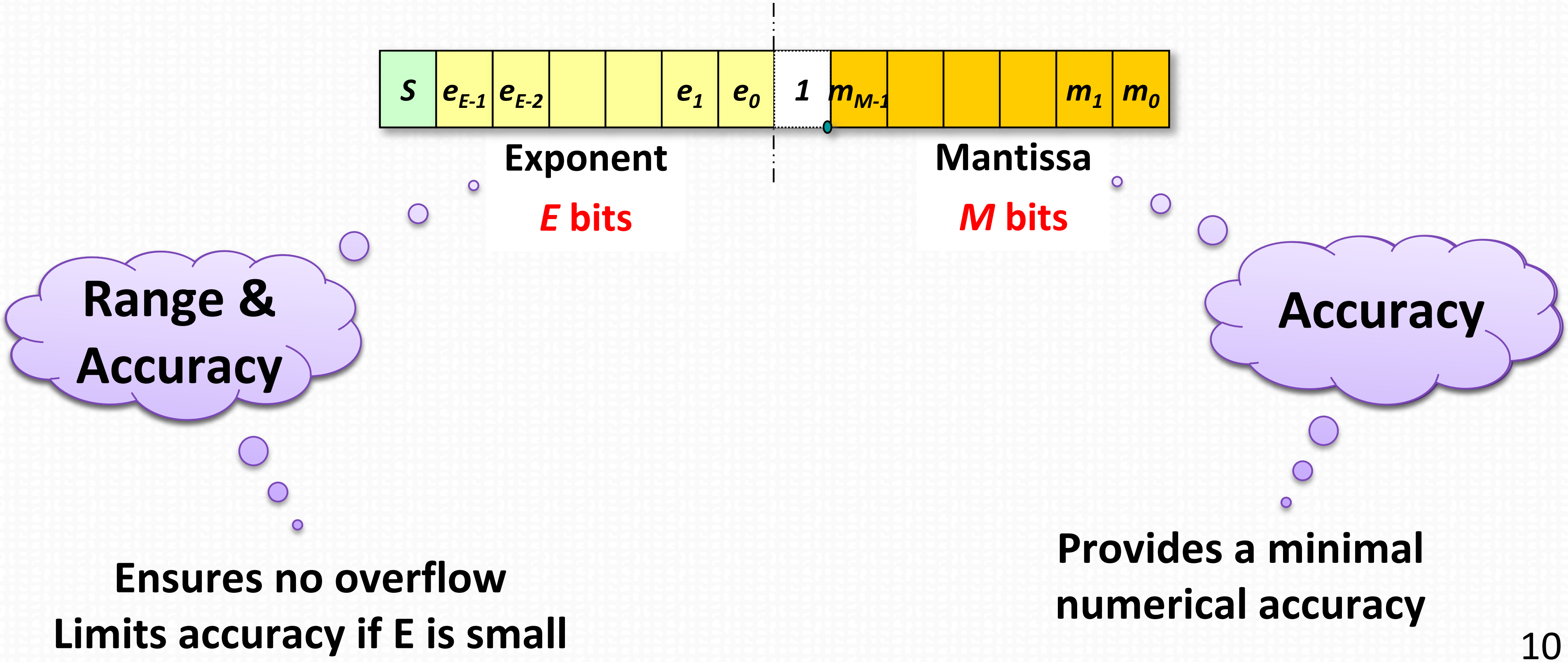


Ensures no overflow
(or limit the overflow occurrence)

Provides a minimal numerical accuracy

Customizing Floating-Point

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



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;
```

ct_float, FloPoCo, ac_float

- ct_float provides comparable (or slightly better) results
  - 16-bit Floating-Point Addition/Subtraction (200MHz)

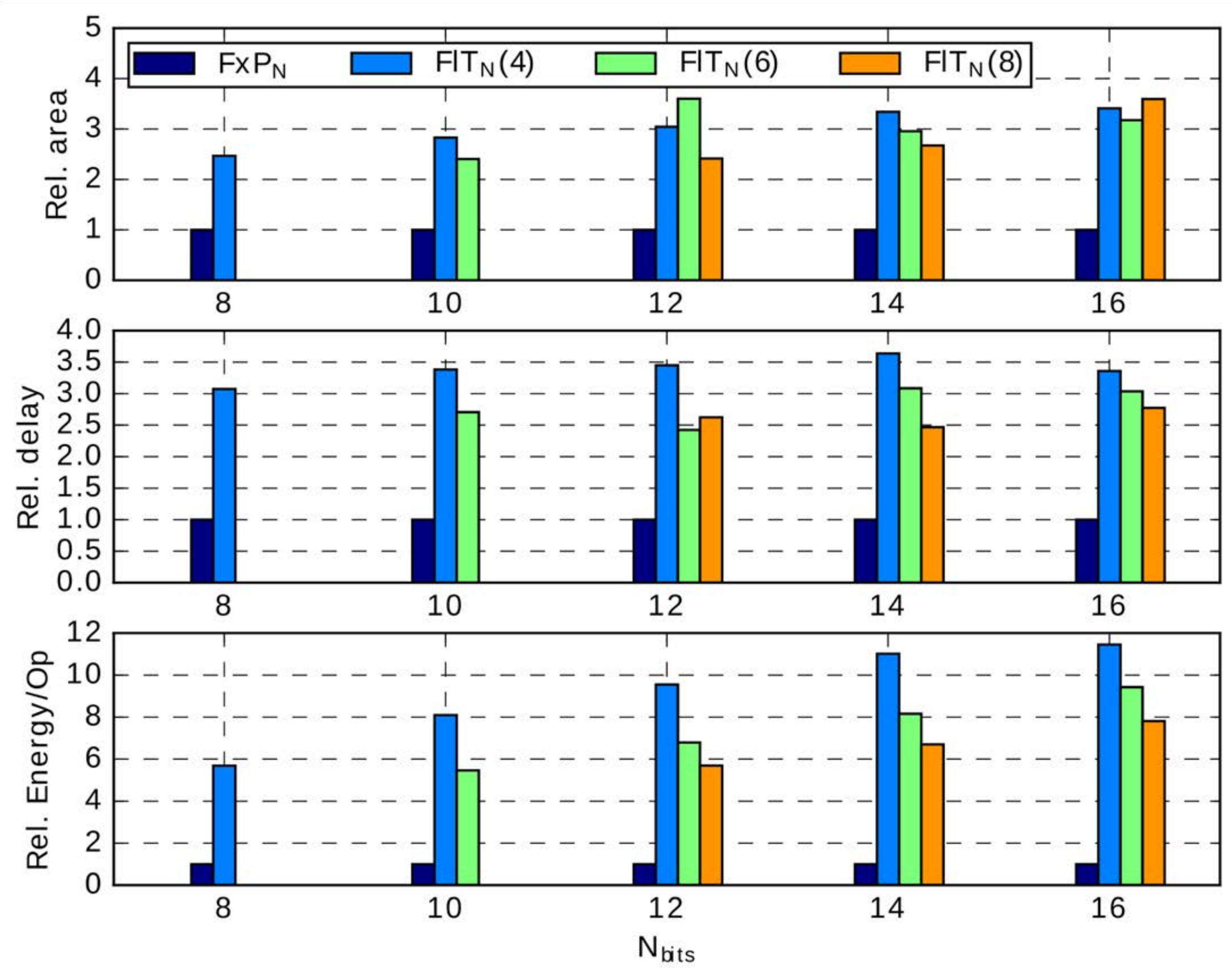
	Area (μm^2)	Critical path (ns)	Total power (mW)	Energy per operation (pJ)
AC_FLOAT	312	1.44	1.84E−1	9.07E−1
CT_FLOAT	318	1.72	2.13E−1	1.05
FLOPOCO	361	2.36	1.84E−1	9.06E−1
CT_FLOAT/AC_FLOAT	**+2.15%**	**+19.4%**	**+15.4%**	**+15.7%**
CT_FLOAT/FLOPOCO	**-11.8%**	**-27.0%**	**+15.7%**	**+15.8%**

  - 16-bit Floating-Point Multiplication (200MHz)

	Area (μm^2)	Critical path (ns)	Total power (mW)	Energy per operation (pJ)
AC_FLOAT	488	1.18	2.15E−1	1.05
CT_FLOAT	389	1.13	1.76E−1	8.59E−1
FLOPOCO	361	1.52	1.34E−1	6.50E−1
CT_FLOAT/AC_FLOAT	**-20.4%**	**-4.24%**	**-18.2%**	**-18.2%**
CT_FLOAT/FLOPOCO	**+7.68%**	**-25.6%**	**+31.7%**	**+32.1%**

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

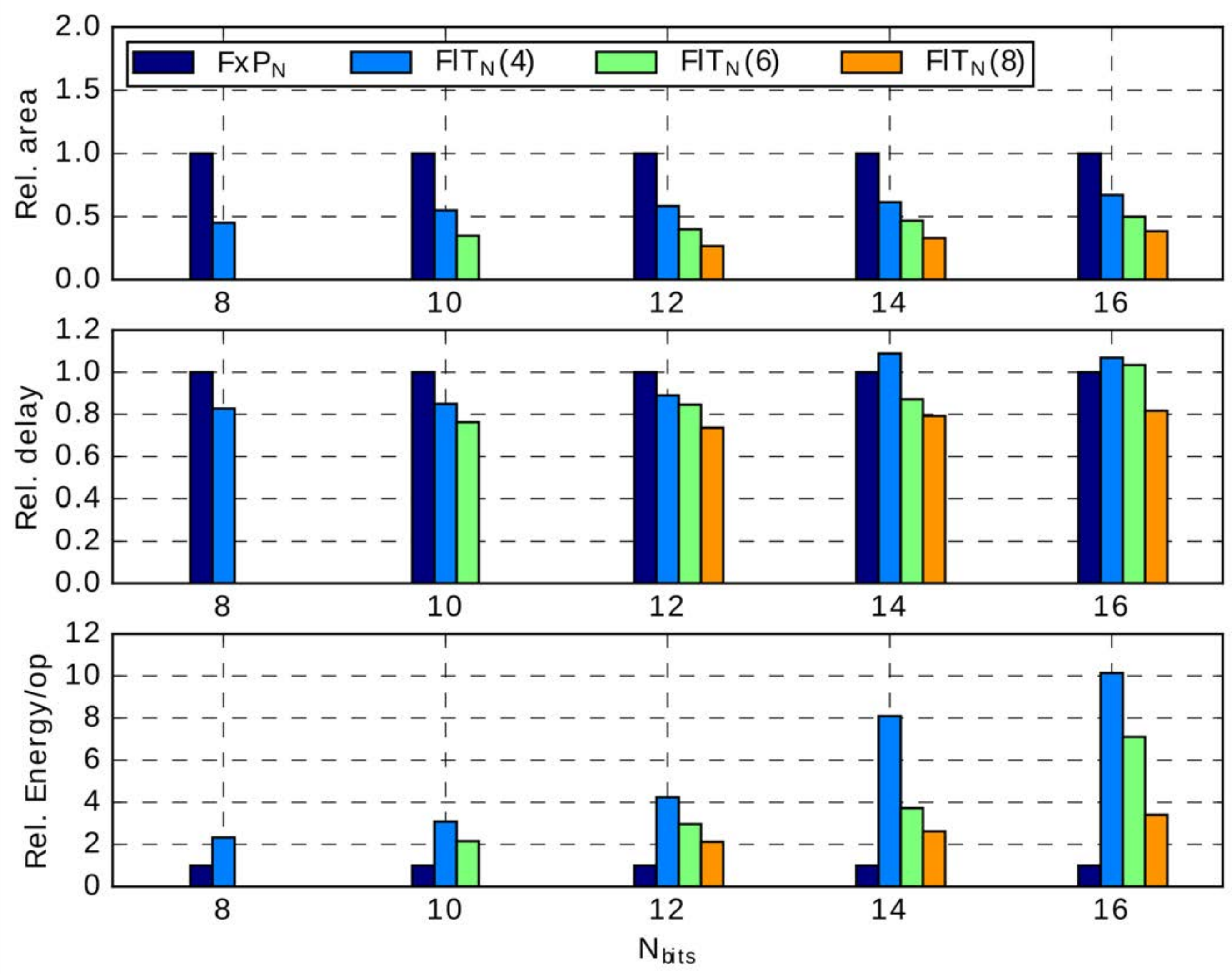


28nm FDSOI technology, Catapult (HLS), Design Compiler, PrimeTime

FxP vs. FlP: Multipliers

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

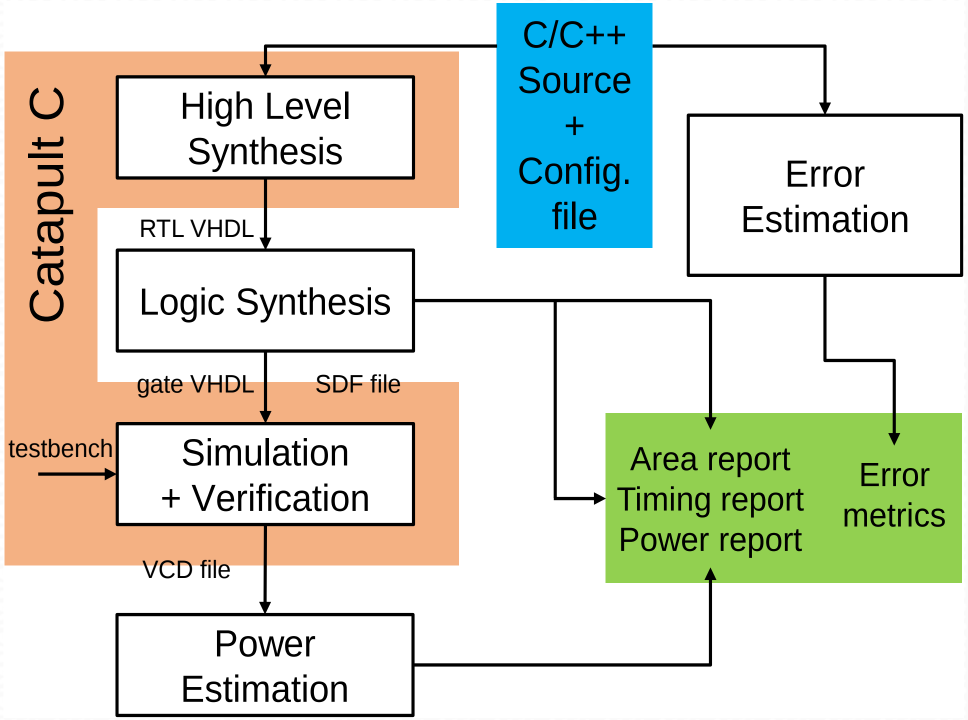
- FlP multipliers are smaller, faster, but consume more energy



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

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



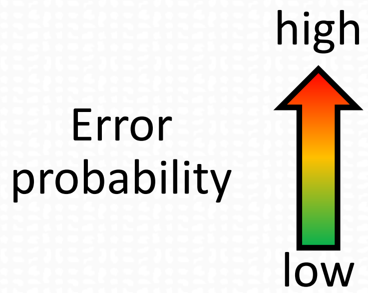
B. Barrois, O. Sentieys, D. Menard, The Hidden Cost of Functional Approximation Against Careful Data Sizing – A Case Study, IEEE/ACM DATE, 2017

Outline

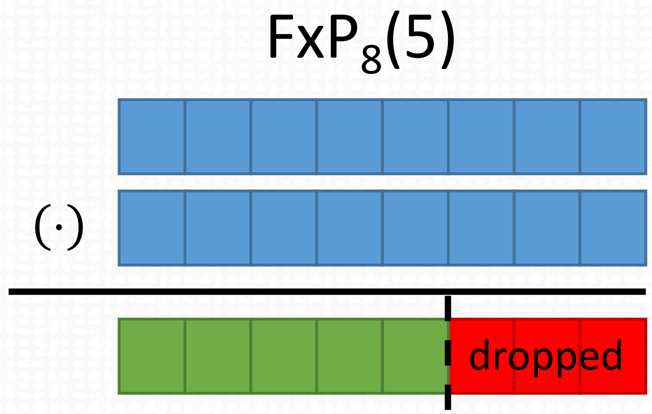
- Motivations for approximate computing
- Number representations
- Approximate operators or careful rounding?
- Operator-level support for approximate computing
- Stochastic computing
- Conclusions

Approximate arithmetic

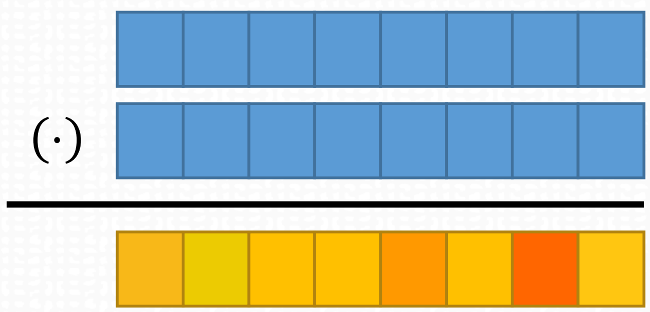
- Comparison of two paradigms



  - Classical fixed-point (FxP) arithmetic
    - Exact integer operators
    - Approximation by rounding the output

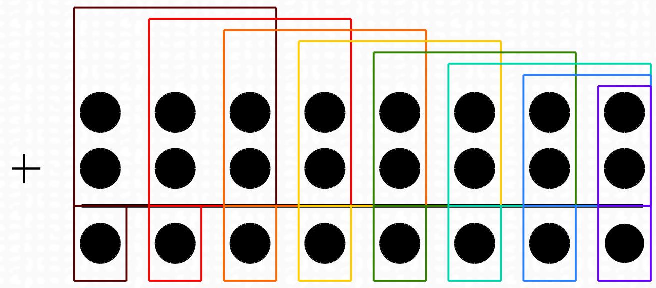


  - Approximate (Apx) integer arithmetic
    - State-of-the-art approximate operators

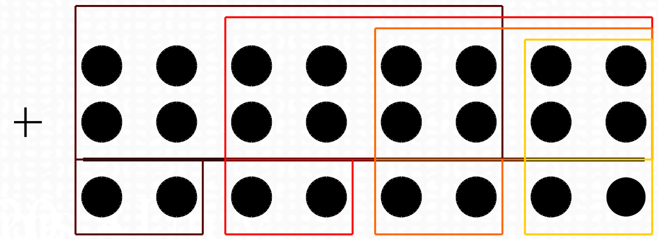


Approximate operators

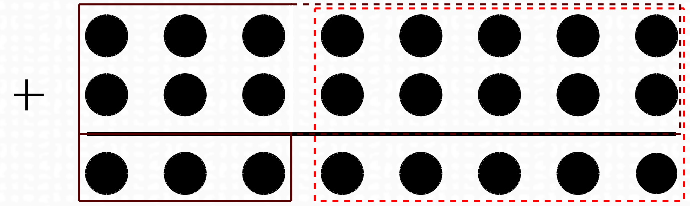
- Adders
  - Almost Correct Adder (ACA)
  - Error-Tolerant Adder IV (ETAIV)
  - Approximate Ripple Carry Adder (RCAApx)
    - 3 possible Full-Adder approximations



$ACA_8(2)$



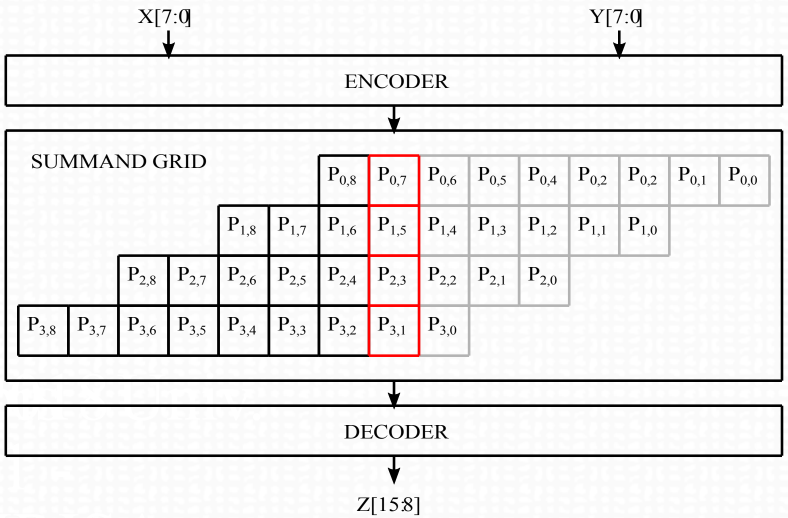
$ETAIV_8(4)$



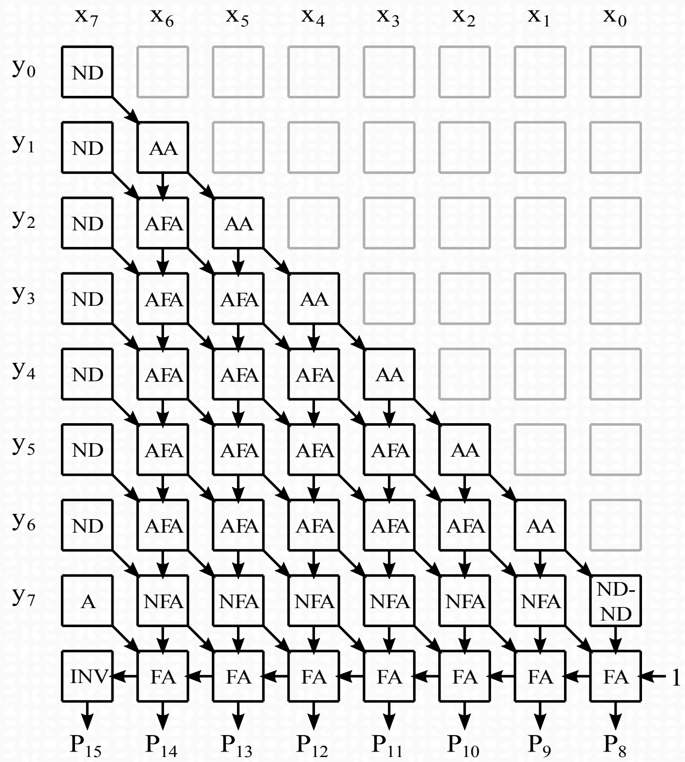
$RCA_{Apx,8}(3)$

Approximate operators

- Fixed-width multipliers
  - Approximate Array Multiplier (AAM)
  - Approximate modified Booth-encoded Multiplier (ABM)



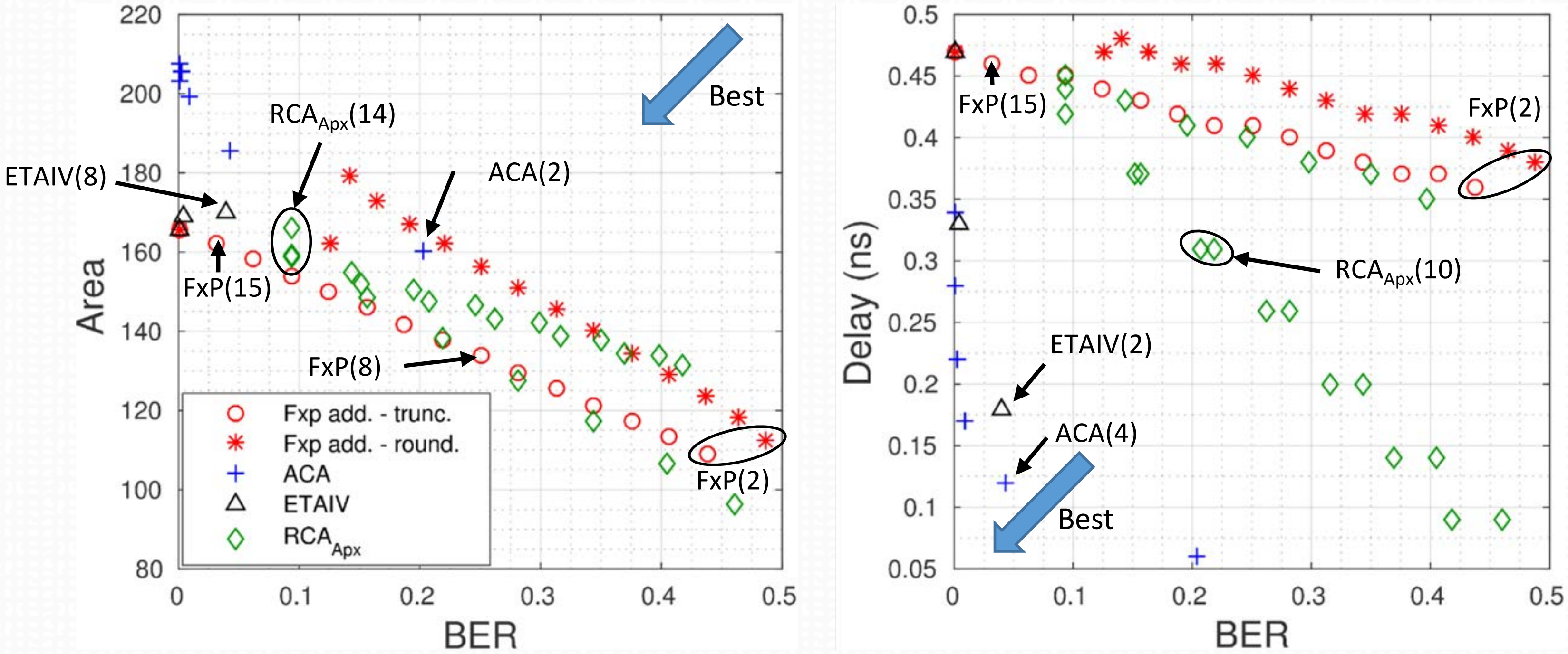
$ABM_8(8)$



$AAM_8(8)$

Approximate or Round?

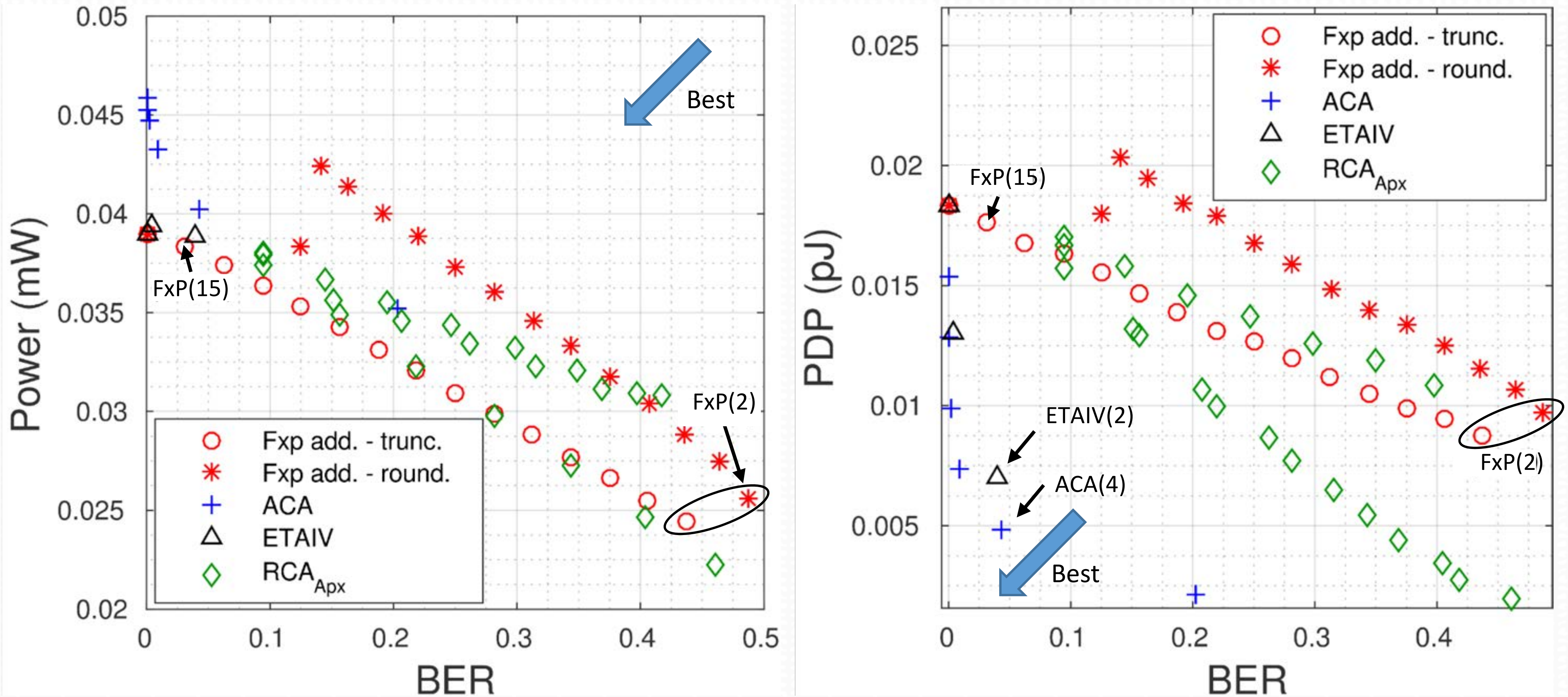
- Results: 16-bit adders



B. Barrois, O. Sentieys, D. Menard, The Hidden Cost of Functional Approximation Against Careful Data Sizing – A Case Study, IEEE/ACM DATE, 2017

Approximate or Round?

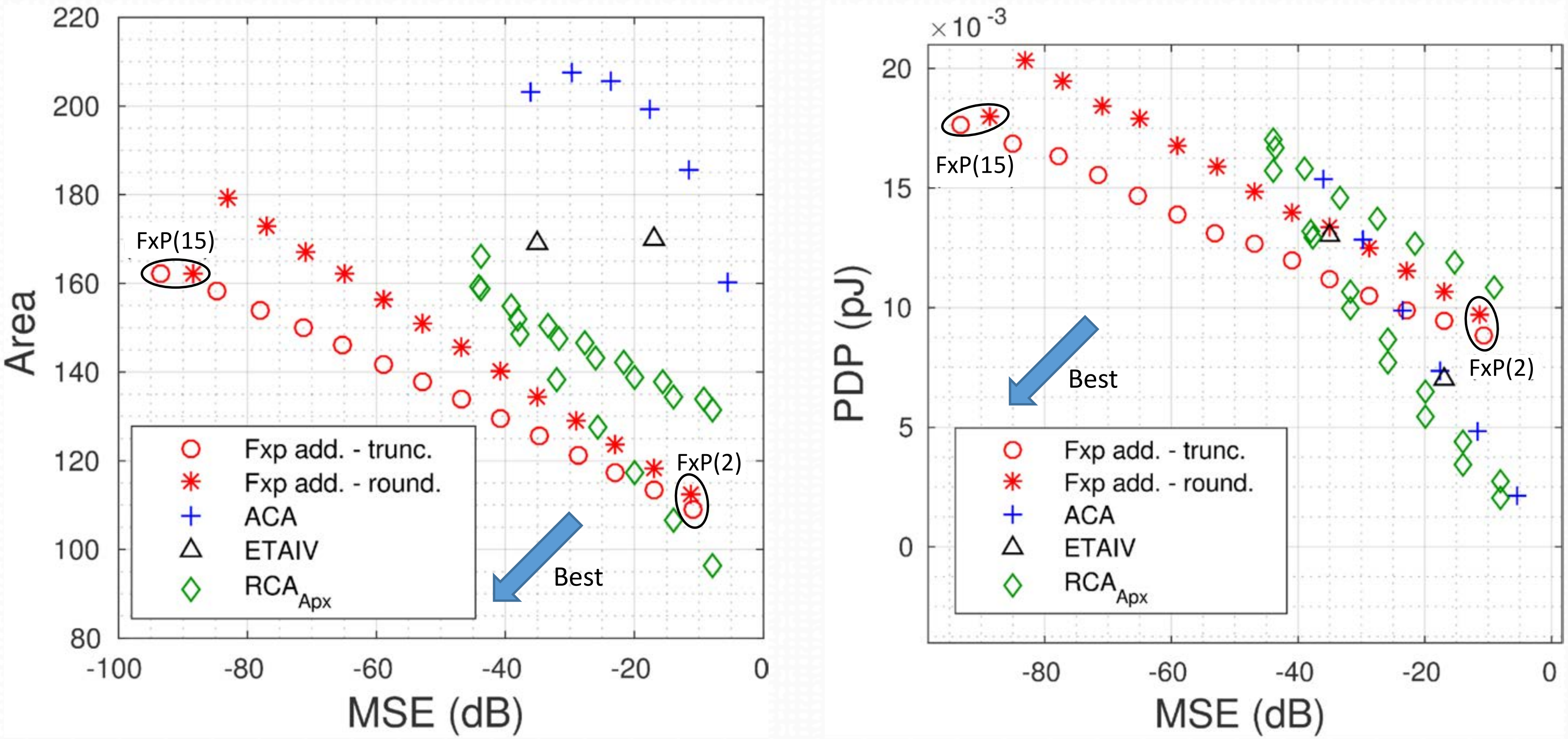
- Results: 16-bit adders



B. Barrois, O. Sentieys, D. Menard, The Hidden Cost of Functional Approximation Against Careful Data Sizing – A Case Study, IEEE/ACM DATE, 2017

Approximate or Round?

- Results: 16-bit adders



B. Barrois, O. Sentieys, D. Menard, The Hidden Cost of Functional Approximation Against Careful Data Sizing – A Case Study, IEEE/ACM DATE, 2017

Approximate or Round?

- Results: Multipliers $16\times16 \rightarrow 16$ bits
  - $MUL_t(16,16)$ is classical exact multiplier with output truncated to 16 bits

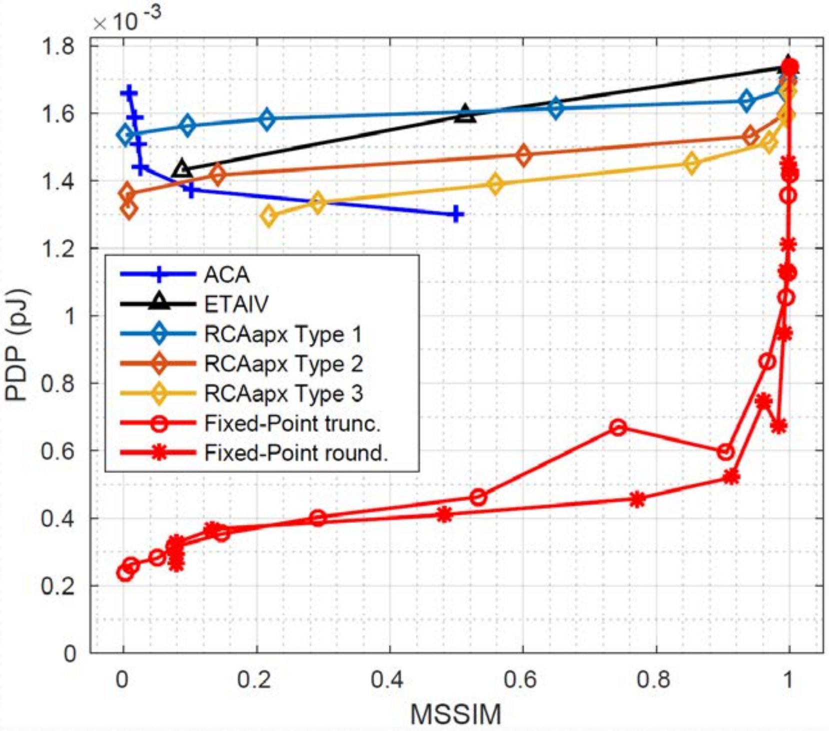
	$\text{FxP}_{t,16}(16)$	$\text{AAM}_{16}(16)$	$\text{ABM}_{16}(16)$
Power (mW)	**0.273**	0.359	0.446
Delay (ns)	0.91	1.23	**0.57**
PDP (pJ)	**0.249**	0.442	0.446
Area (μm^2)	805.2	**665.5**	879.5
BER (%)	**23.4**	27.7	27.9
MSE (dB)	**-89.1**	-87.9	-9.63

Performance of FxP and AO multipliers

B. Barrois, O. Sentieys, D. Menard, The Hidden Cost of Functional Approximation Against Careful Data Sizing – A Case Study, IEEE/ACM DATE, 2017

Approximate or Round?

- Results on applications
  - JPEG, HEVC, K-Means



Adders – Apx DCT cost in JPEG encoding

	MSSIM	Adder Energy (pJ)	Min. Mult. Energy (pJ)	Total Energy (pJ)
$ADD_t(16, 10)$	99.29%	1.39E−2	4.39E−2	0.898
ACA(16, 12)	96.45%	1.54E−2	2.49E−1	4.20
ETAIV(16, 4)	98.02%	1.30E−2	2.49E−1	4.17
$RCA_{Apx}(16, 6, 3)$	99.67%	1.00E−2	2.49E−1	4.12

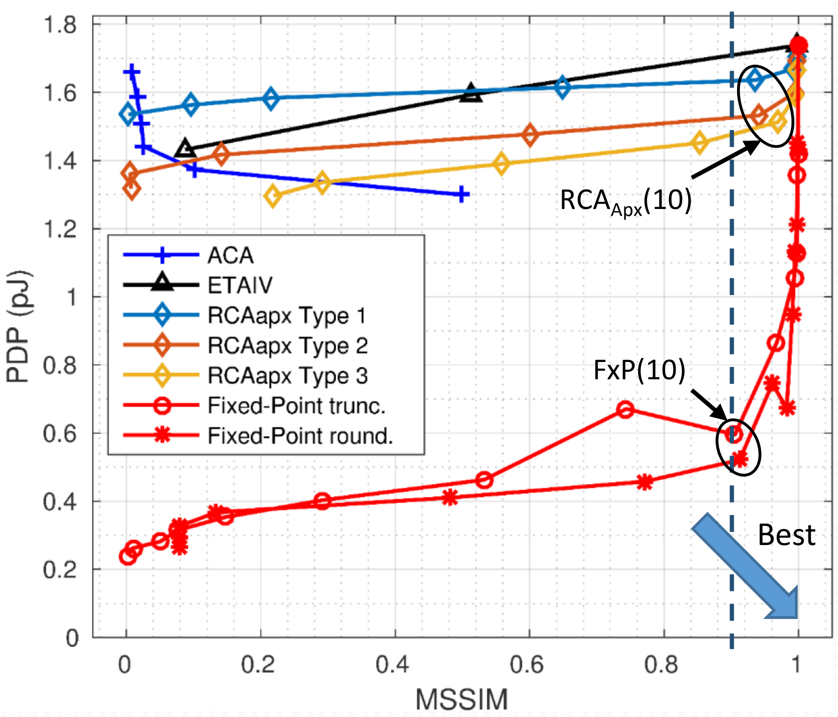
Adders – cost in HEVC filter

	Success Rate	Multiplier Energy (pJ)	Min. Adder Energy (pJ)	Total Energy (pJ)
$MUL_t(16, 16)$	99.84%	2.49E−1	1.83E−2	5.15E−1
AAM(16)	99.43%	4.42E−1	1.83E−2	9.02E−1
ABM(16)	10.27%	2.54E−1	1.83E−2	5.27E−1
$MUL_t(16, 4)$	10.87%	2.04E−1	1.24E−3	4.09E−1

Multipliers – cost of distance computation in K-Means algorithm

Approximate or Round?

- Results: DCT in JPEG Encoding – 90% effort



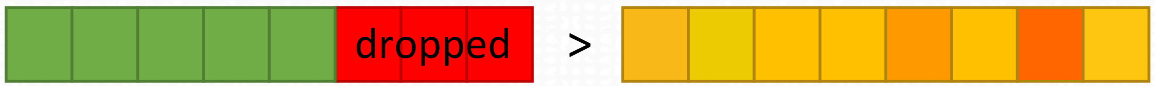
$FxP_{t,16}(16)$
MSSIM = 0.9981
PDP = 1.73 pJ

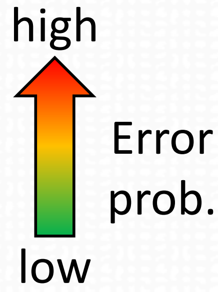
$AAM_{16}(16)$
MSSIM = 0.9981
PDP = 2.71 pJ

$ABM_{16}(16)$
MSSIM = 0.8579
PDP = 2.72 pJ

Conclusion (Apx. or Round?)

- Datasize reduction gives better results than operator-level approximation
- High error entropy is not energy efficient





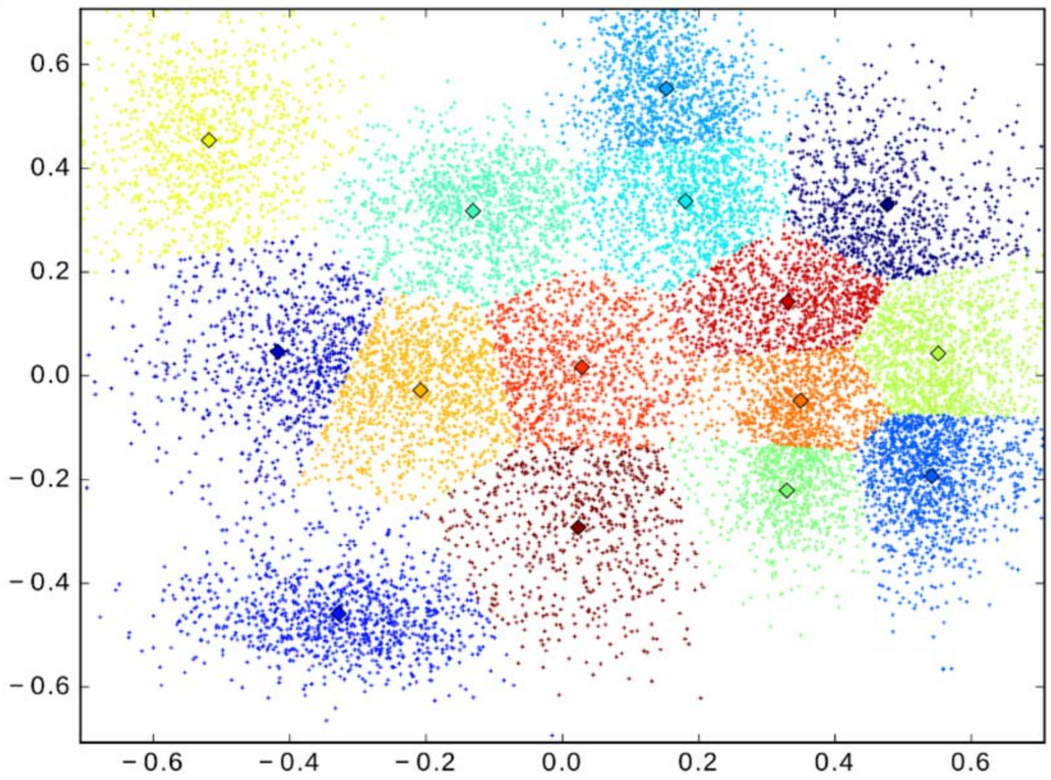
  - True for processing datapath
  - Should be emphasized when considering data storage and transportation
- Approximate operators could be suitable for fixed-width datapath (e.g. CPU)

Outline

- Motivations for approximate computing
- Number representations
- Approximate operators or careful rounding?
- Operator-level support for approximate computing
  – K-Means Clustering, FFT
  – Approximate deep learning
- Stochastic computing
- Conclusions

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



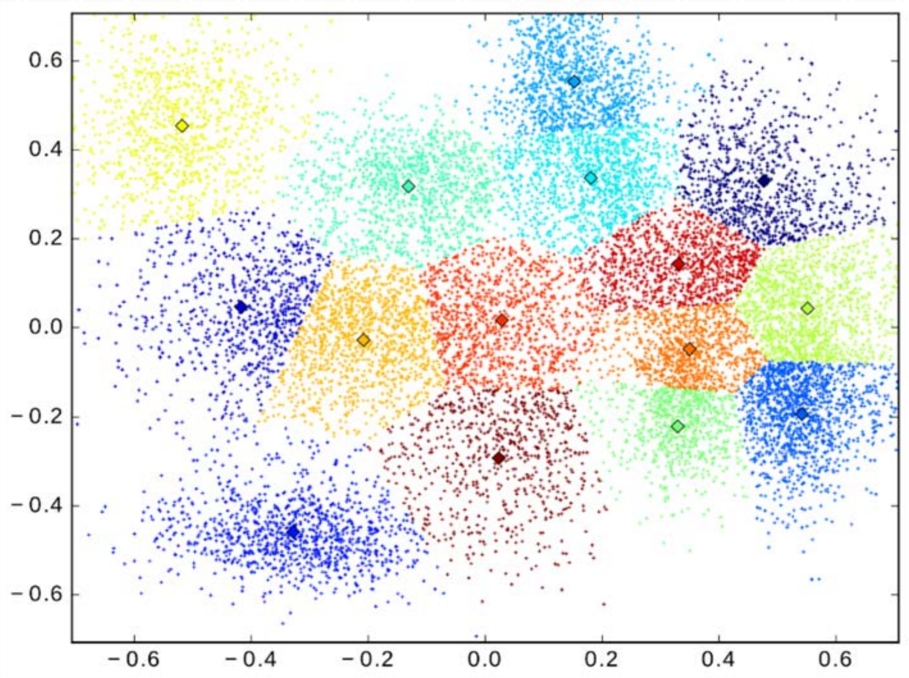
- Finding the set of clusters $S = \{S_i\}_{i \in [0, k-1]}$

  satisfying $$\arg\min_S \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|^2$$

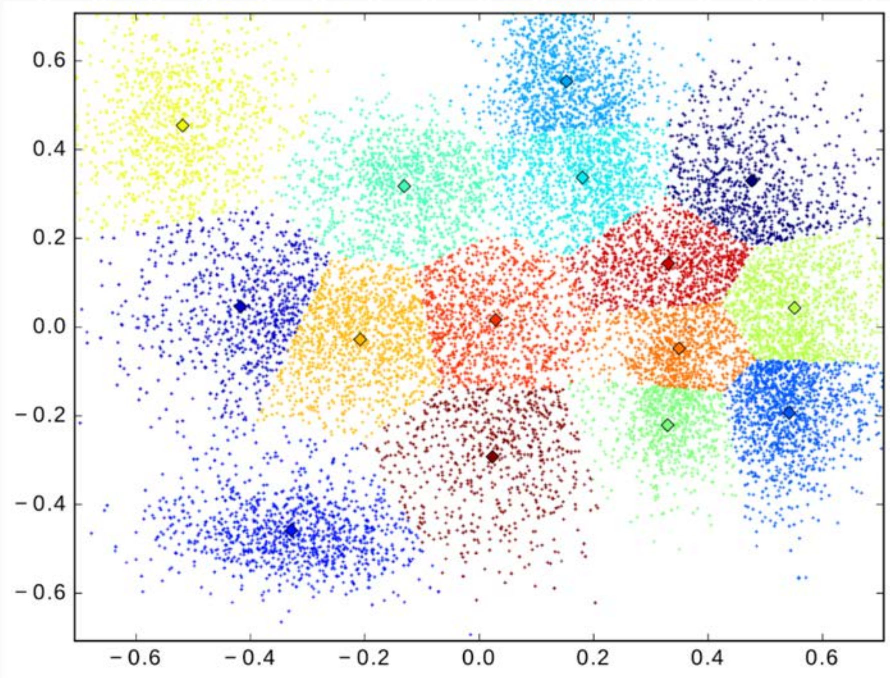
  is NP-hard

Approximate K-Means Clustering

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

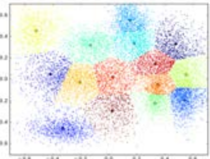


Reference: double

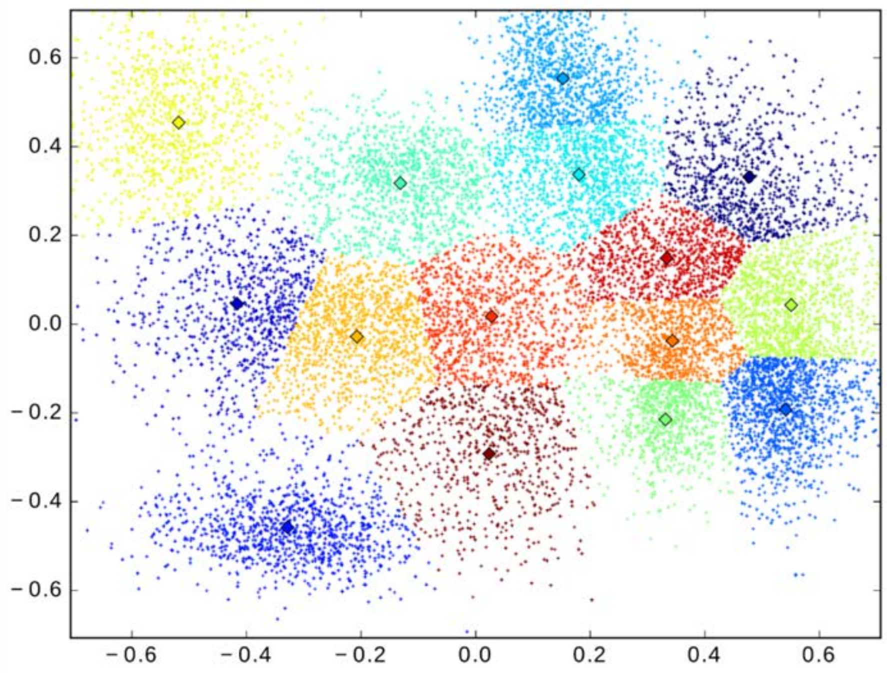


Floating-point: ct_float_{16}
5-bit exponent
11-bit mantissa

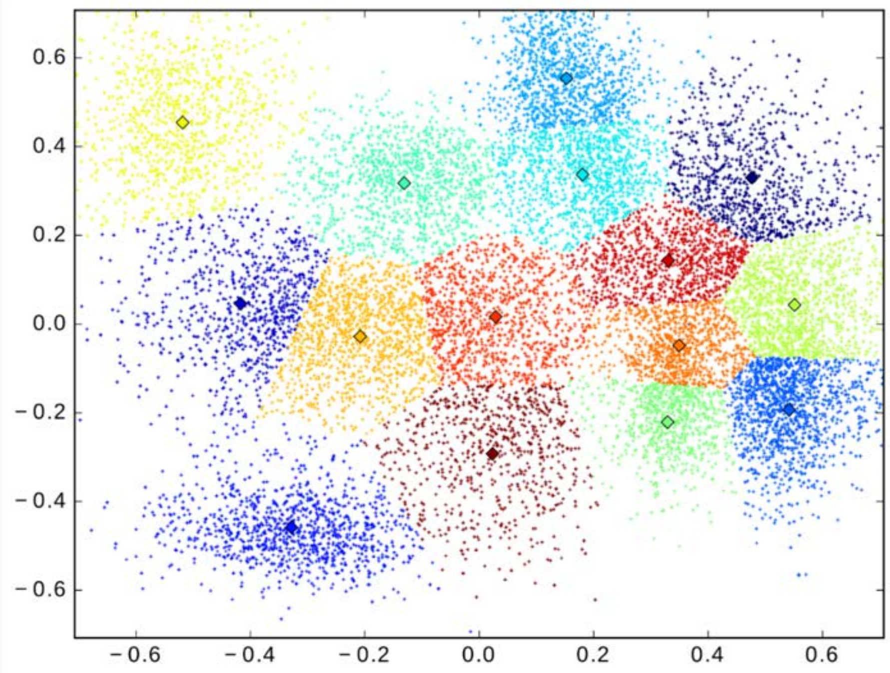
Approximate K-Means Clustering



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



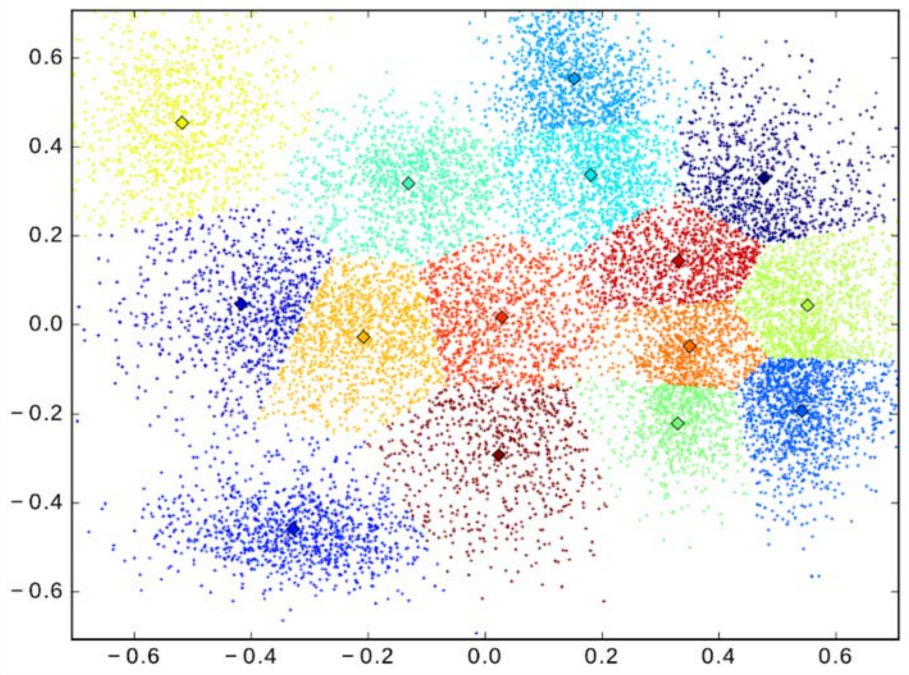
Fixed-Point: ac_fixed_{16}
3-bit integer part
13-bit fractional part



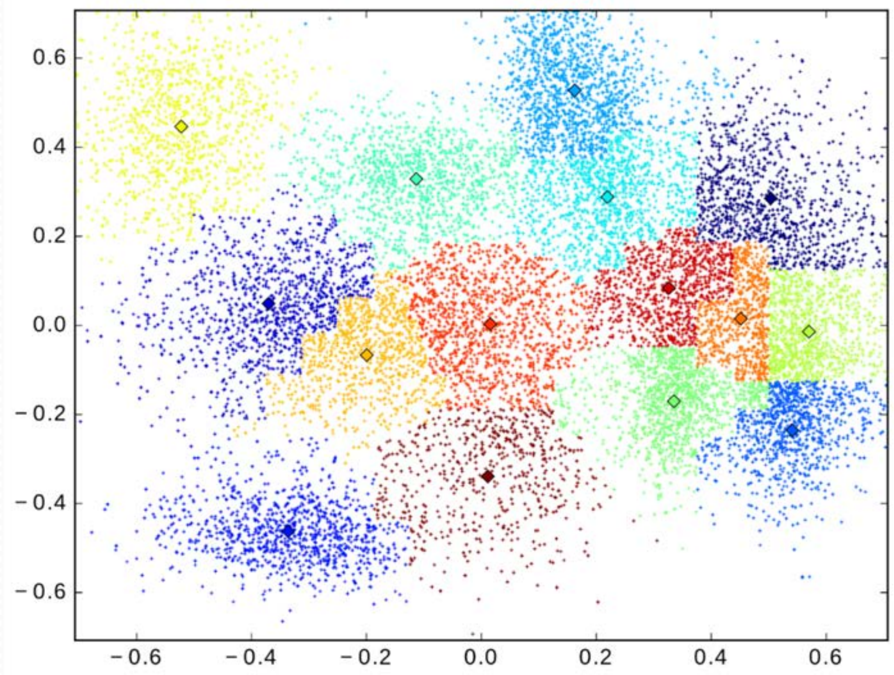
Floating-point: ct_float_{16}
5-bit exponent
11-bit mantissa

Approximate K-Means Clustering

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

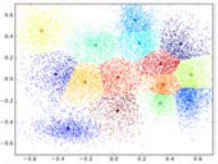


Reference: double

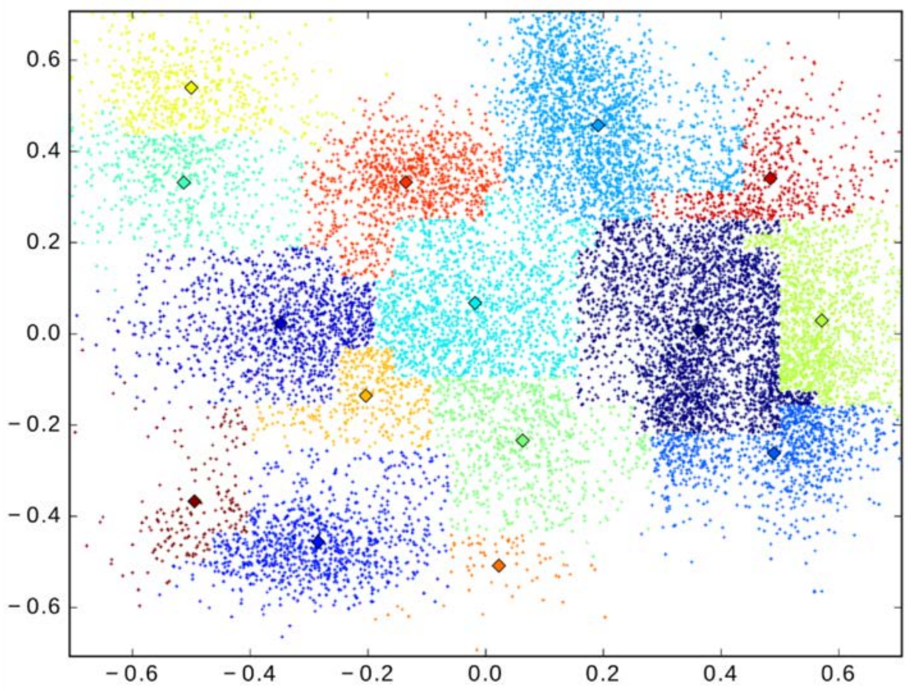


Floating-Point: ct_float_8
5-bit exponent
3-bit mantissa

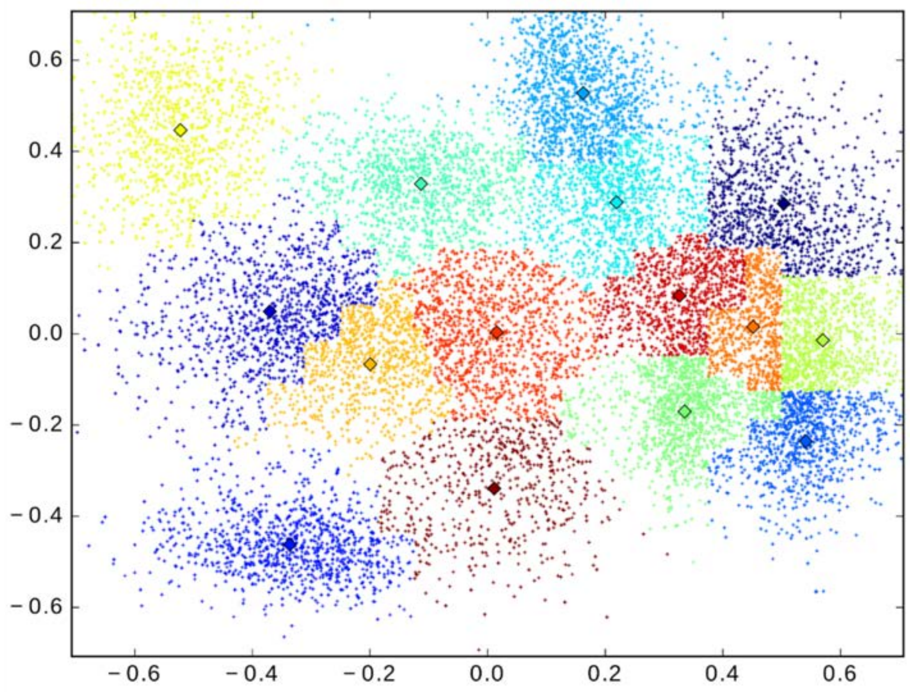
Approximate K-Means Clustering



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



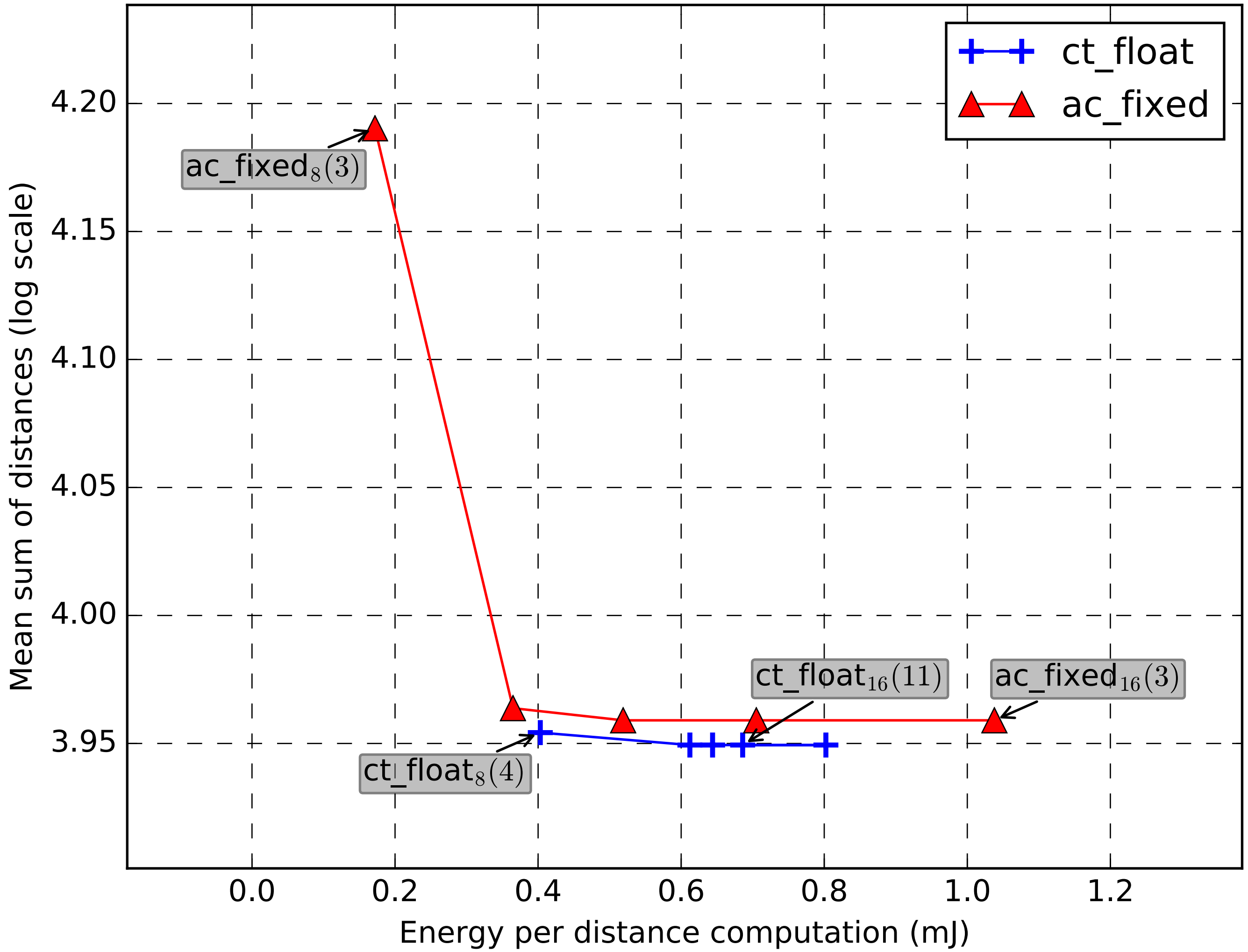
Fixed-Point: ac_fixed_8
3-bit integer part
5-bit fractional part



Floating-Point: ct_float_8
5-bit exponent
3-bit mantissa

Energy versus Mean Sum of Distances

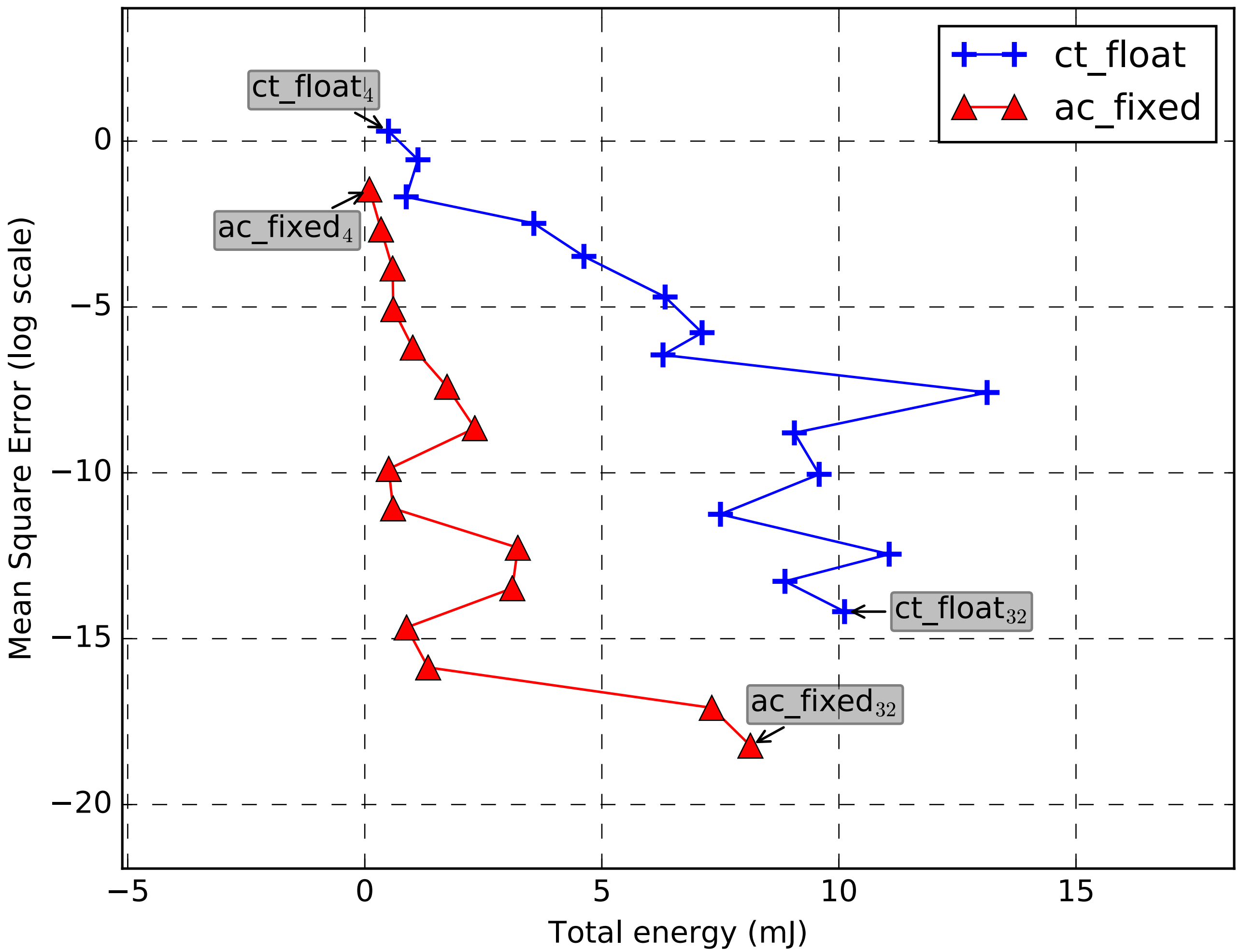
- Average energy consumed by K-means algorithm
- Stopping condition: 10^{-4}



B. Barrois, O. Sentieys, Customizing Fixed-Point and Floating-Point Arithmetic - A Case Study in K-Means Clustering, IEEE SiPS, 2017.

Energy vs. Error: FFT

- FxP performs always better (5×) than FlP

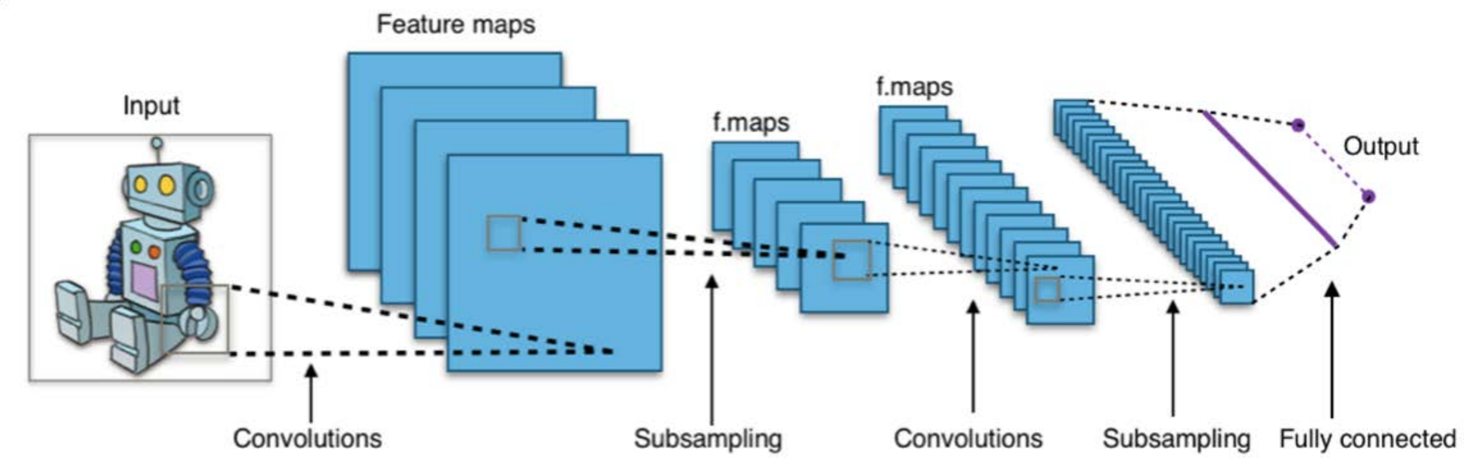


Conclusions (FlP vs. FxP)

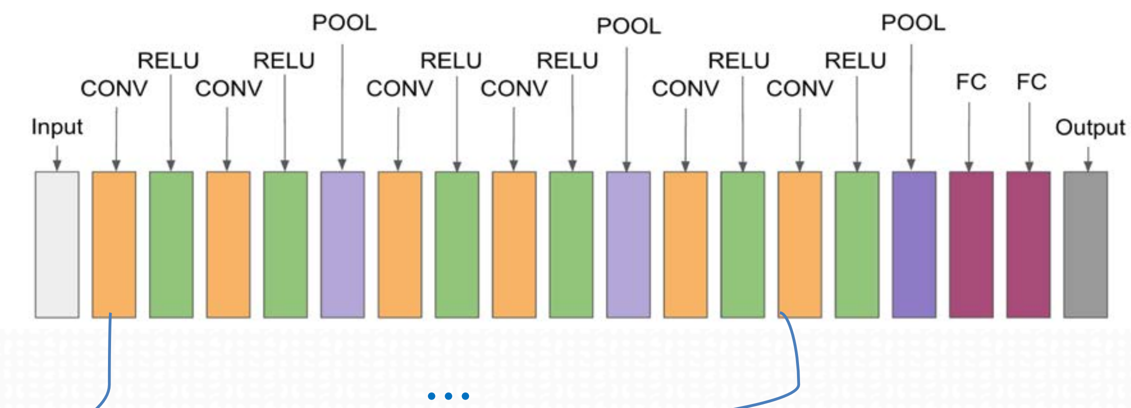
- Slower increase of errors for floating-point
  - Small floating-point (e.g. 8-bit) could provide better error rate/energy ratio
    - 8-bit FlP is still effective for K-means clustering
- Choice FlP vs. FxP is not obvious
  - Application-dependent
  - Certainly requires static/runtime analysis
- Perspectives
  - Custom exponent bias in *ct_float*
  - Towards an automatic optimizing compiler considering both FxP and FlP representations

Deep Convolutional Neural Networks

- General organization



- Layers

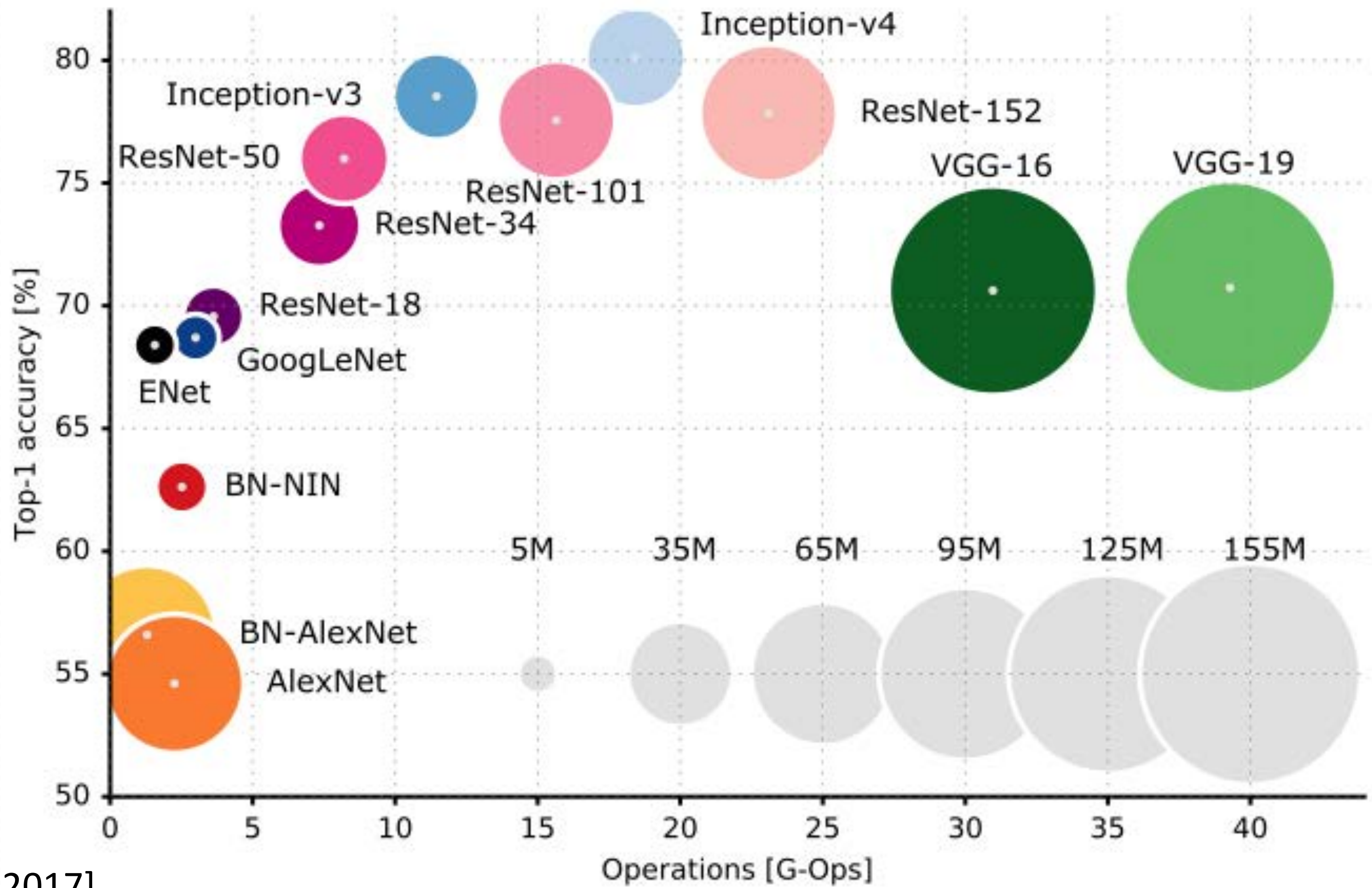



```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
   L:  output_fm[to][row][col] +=
          weights[to][ti][i][j]*
          input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

[Motamedi et al., 2016]

Complexity of Deep CNNs

- 10-30 GOPS
  - Mainly convolutions
- 10-200 MB
  - Fully-connected layers



[Canziani *et al.*, 2017]

Resilience of ANN

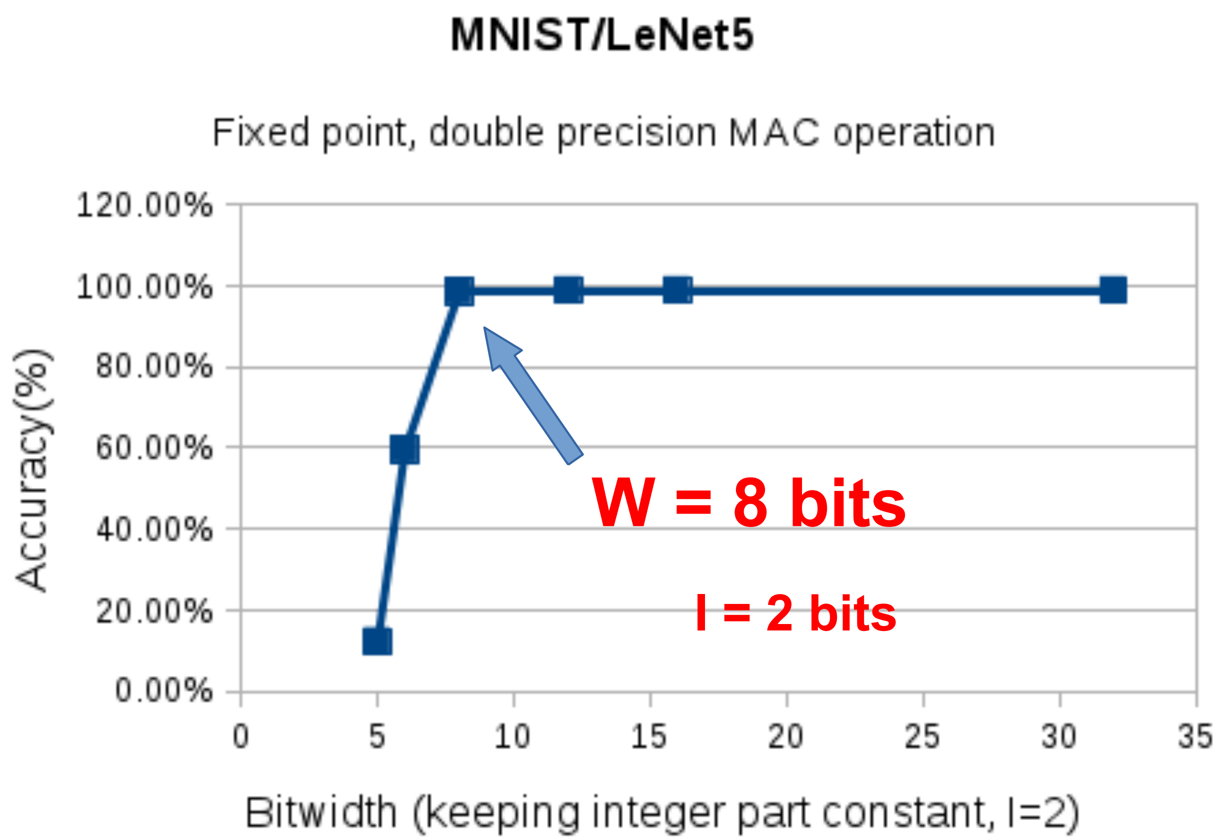
Aoccdrnig to a rscheearch at Cmabrigde Uinervtisy, it deosn't mttaer in waht oredr the ltteers in a wrod are, the olny iprmoatnt tihng is taht the frist and lsat ltteer be at the rghit pclae. And we spnet hlaf our lfie larennig how to splel wrods. Amzanig, no!

[O. Temam, ISCA10]

- Our biological neurons are fault tolerant to computing errors and noisy inputs
- Quantization of parameters and computations provides benefits in throughput, energy, storage

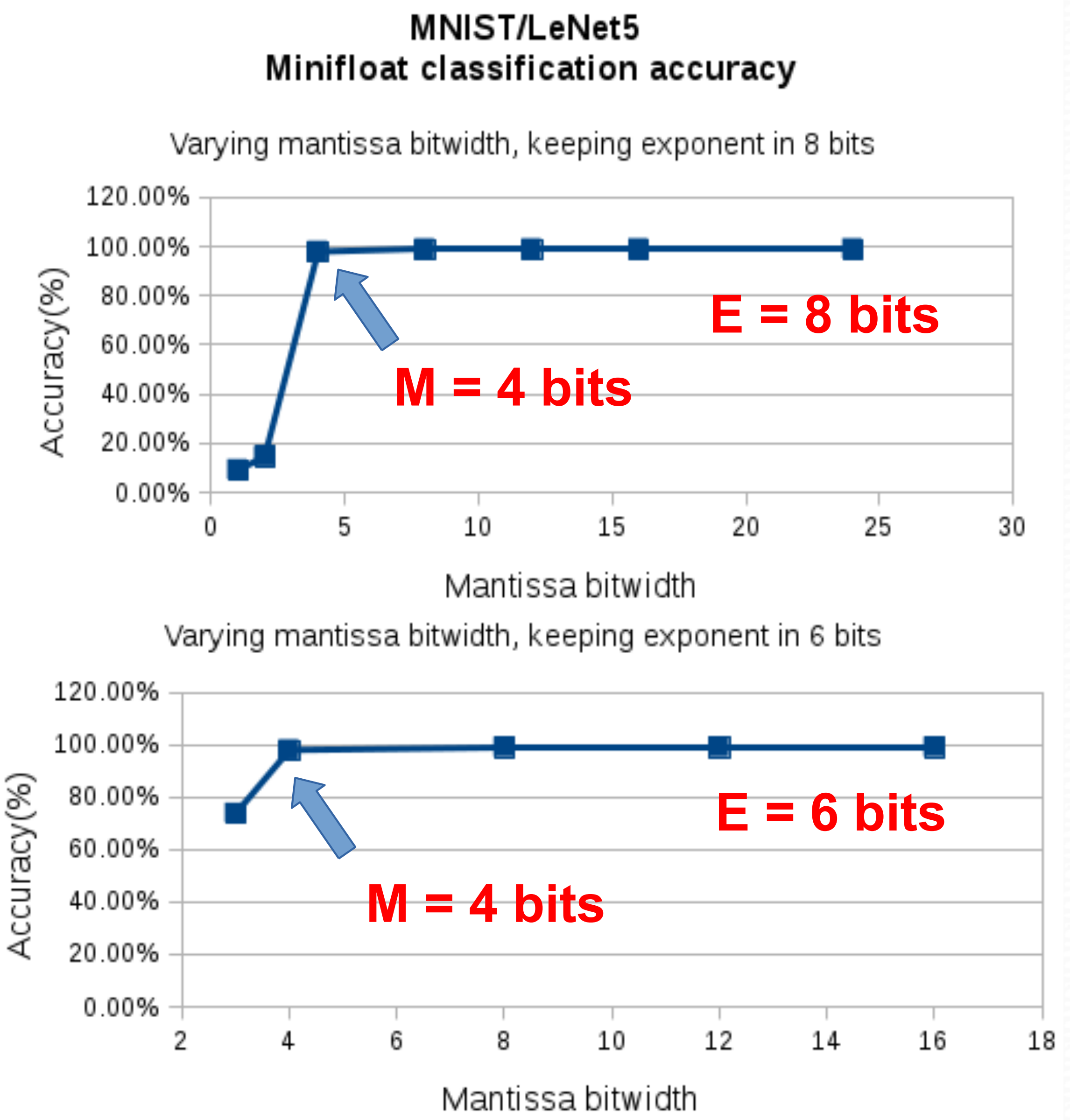
Approximate CNNs: Accuracy

- 10k images, MNIST/Lenet5
- Single/Double-Precision Fixed-Point



Approximate CNNs: Accuracy

- 10k images, MNIST/Lenet
- Custom Floating-Point
- 10-bit FxP or FlP keeps accuracy near reference
- Better results would be achieved with longer training and fine tuning



Outline

- Motivations for approximate computing
- Number representations
- Operator-level support for approximate computing
- Approximate operators or careful rounding?
- Stochastic computing
  - What is a stochastic number?
  - Basic operators
  - Stream correlation
  - Examples
    - Digital filters
    - Image processing
- Conclusions

A Strange Way to Represent Numbers

- Stochastic numbers are represented as a Bernoulli random process
  - p is coded as a finite sequence of independent random variables $x_i \in \{0, 1\}$, with $P(x_i=1) = p$

- Unipolar: $p \in [0, 1]$
  - stream of N bits *X=<x0, x1,…,xN-1>*

  <00010100> = 1/4

  <0010010010000001> = 1/4
  - N_1 ones, N-N_1 zeros: $p=N_1/N$

- Bipolar: $p \in [-1, 1]$, $2.P(x_i)-1=p$

  <00010100> = -1/2

Stochastic Computing

- Uses Stochastic Number representation
- Uses conventional logic circuits to implement arithmetic operations with SNs
  - Realized by simple logic circuits
- SC provides massive parallelism
- SN is intrinsically error tolerant
- Only suitable for low-precision (~5-6 bits)
- High processing latency (e.g. 128-bit streams)

Numerical Accuracy of SNs

- Estimation of *p* out of the *N*-bit stream *X*

$$\hat{p} = \frac{N_1}{N}$$

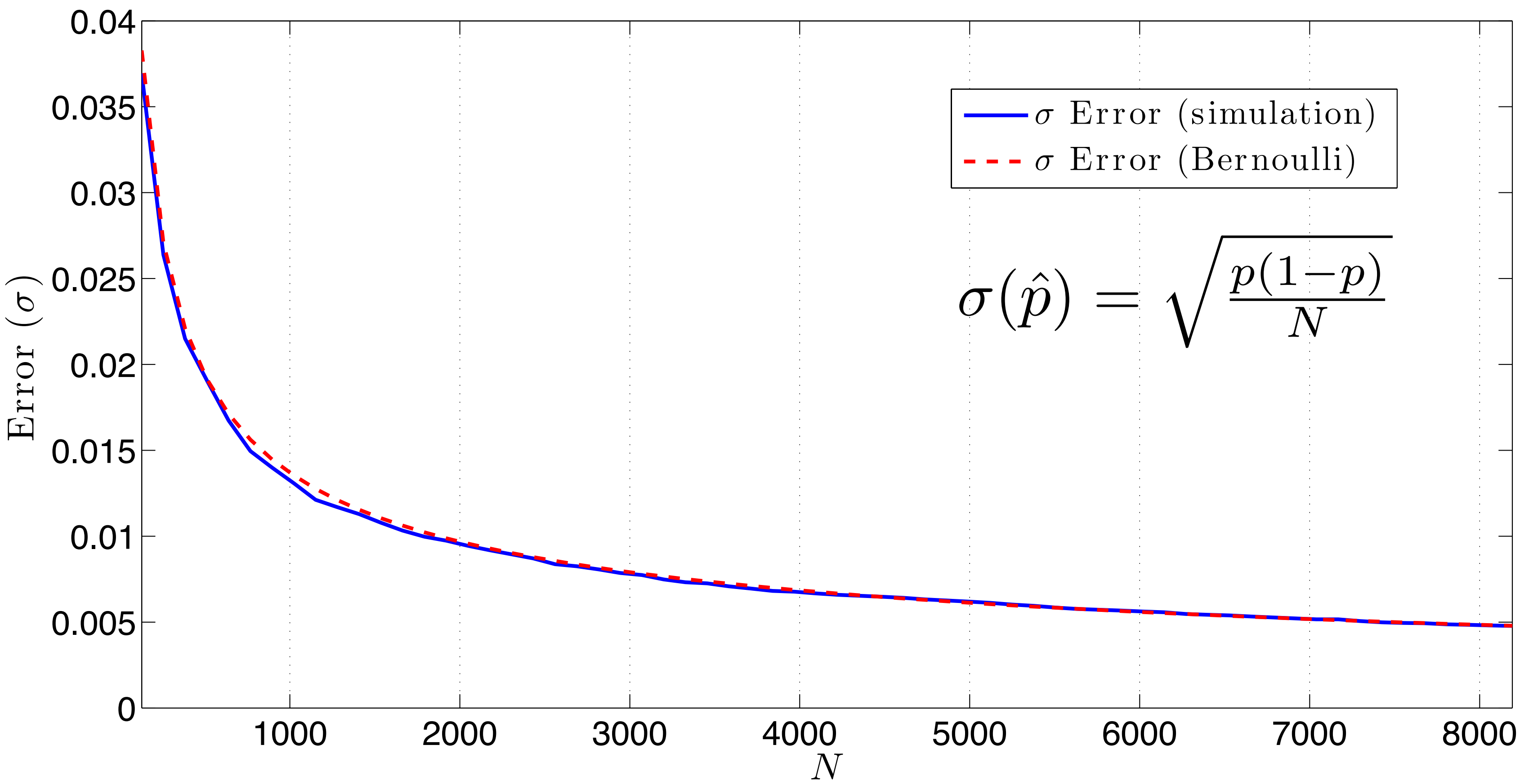
$$E(\hat{p}) = p$$

$$\sigma(\hat{p}) = \sqrt{\frac{p(1-p)}{N}}$$

  - Binomial distribution
- Accuracy in estimation of p increases as square root of *N* (computation time)
- Example: *N=256*
  - Possible values of *p* ∈ *{0, 1/256, 2/256,…,255/256,1}*
  - Accuracy
    - minimum for *p={0,1}*, maximum for *p=0.5*
    - *p=0.75: σ=0.027 (≈5.2 bits)* *(1/256=0.0039)*

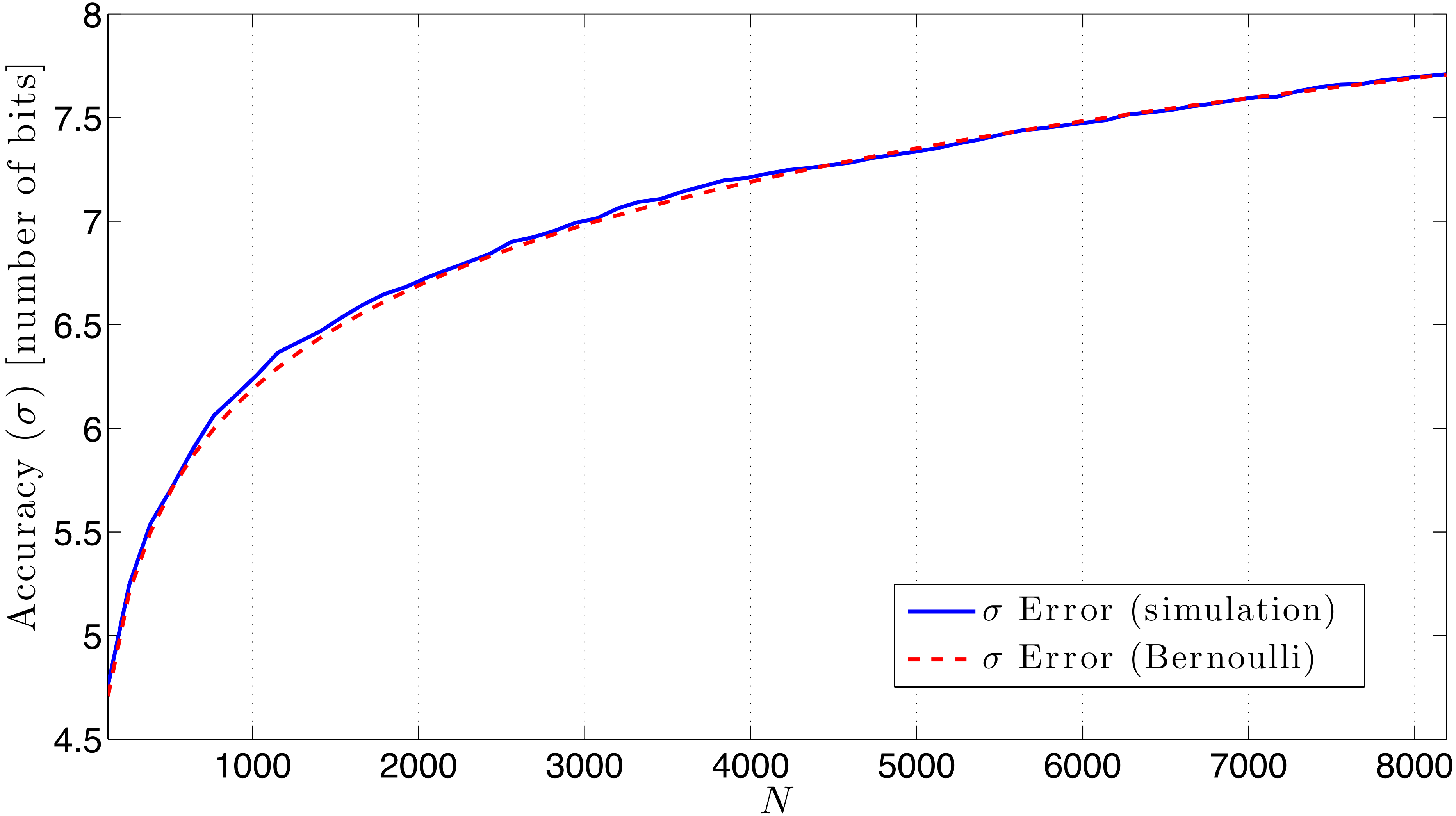
Numerical Accuracy of SNs

- p=0.75, N=128..8192
- σ of error: simulation and analytical



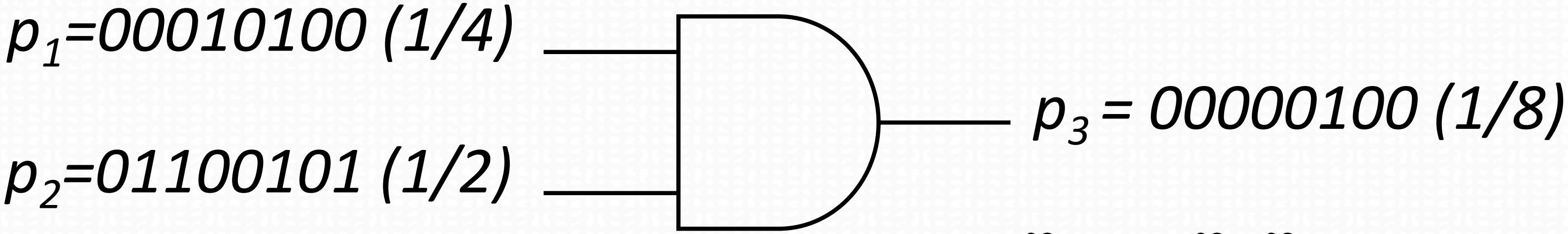
Numerical Accuracy of SNs

- p=0.75, N=128..8192



Basic Arithmetic Operators

- Unsigned multiplication

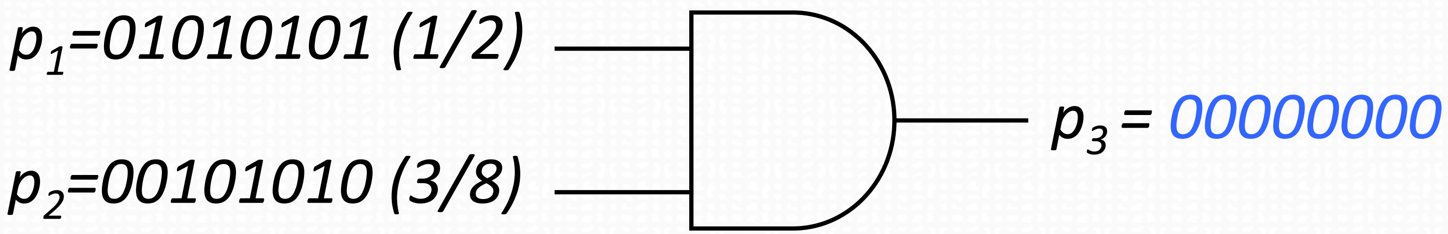


$$p_3 = p_1 p_2$$
$$E(\hat{p_3}) = E(\hat{p_2}).E(\hat{p_1})$$

- Nice! but for real cases, accuracy is reduced
- and p_3 must be longer
- and true only for uncorrelated p_i

Correlation further Reduces Accuracy

- Correlation among bit streams implies reduced accuracy



[Alaghi et al., ACM TECS 2013]

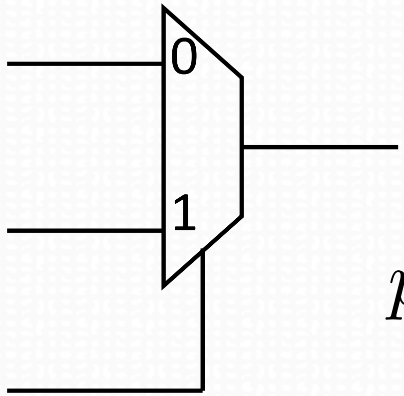
Basic Arithmetic Operators

- Addition (stochastic weighted summer)

p_1=00010100 (1/4)

p_2=01100110 (1/2)

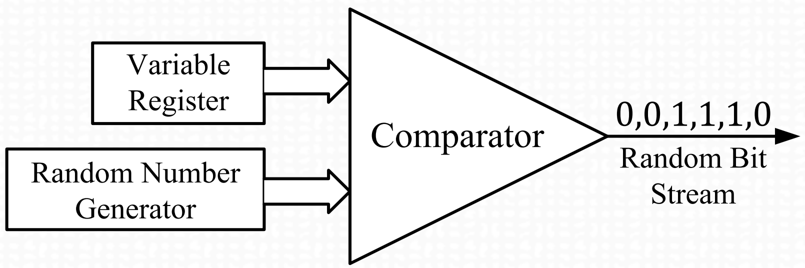
p_0=01011010 (1/2)



p_3 = 01000110 (3/8)

$$p_3 = p_0 p_1 + (1 - p_0) p_2$$

- Stochastic Number Generation
  - E.g.



Error Tolerance

- Conventional computing

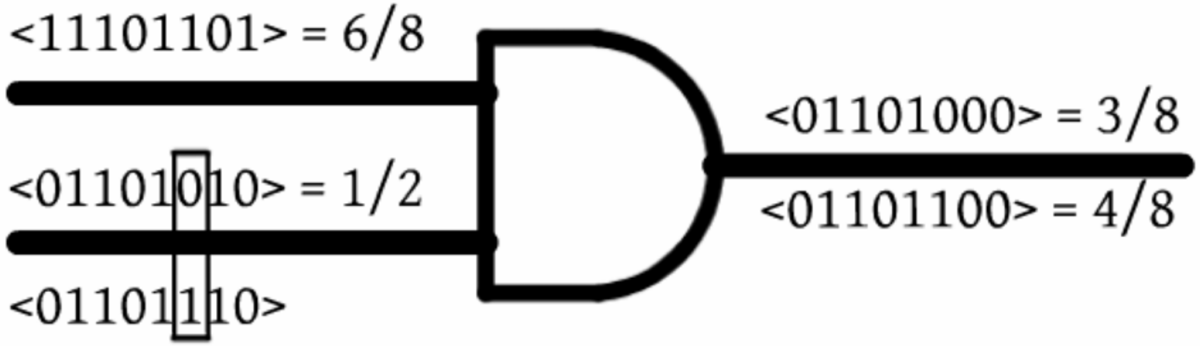
6/8 1/2 3/8

.110 x .100 = .011

.010 x .100 = .001

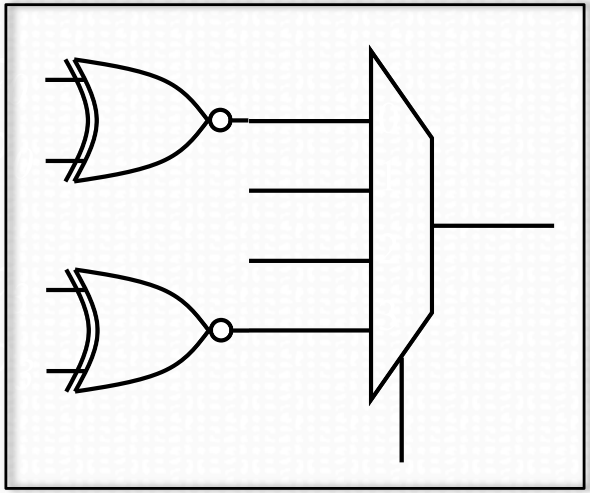
1/8

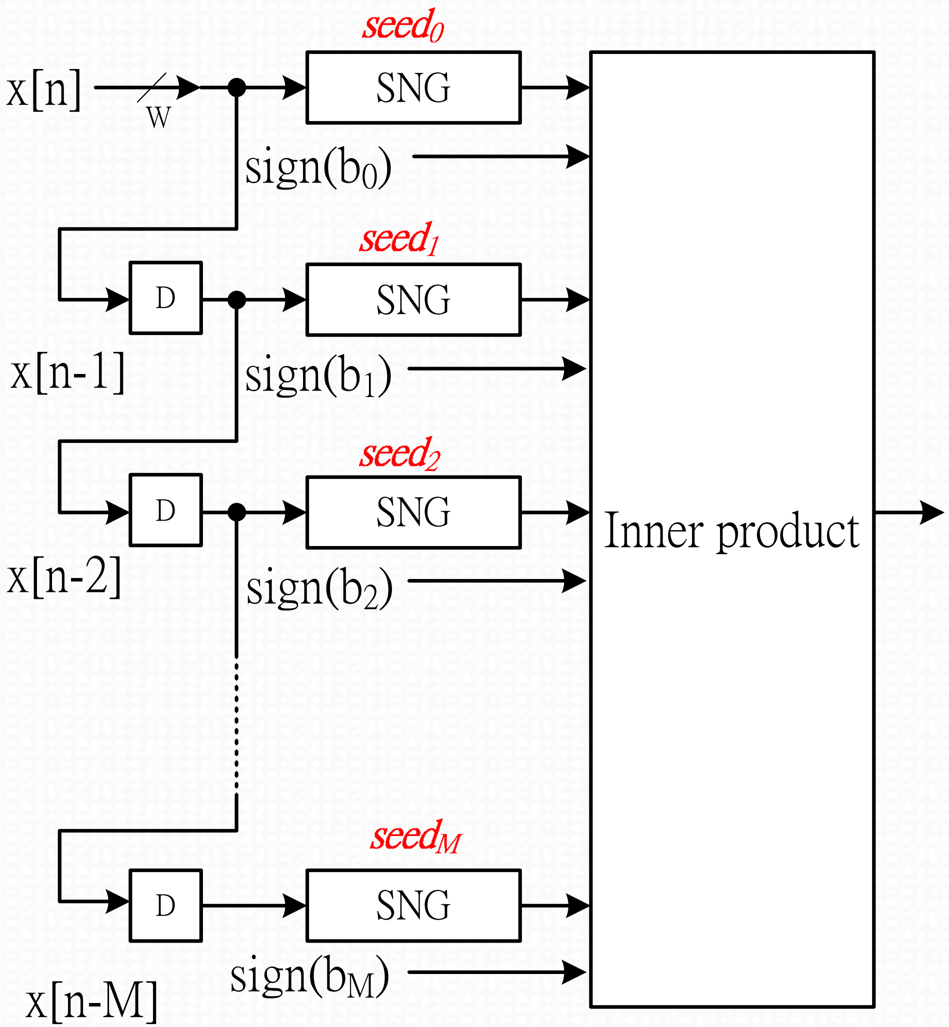
- Stochastic computing



Digital Filters

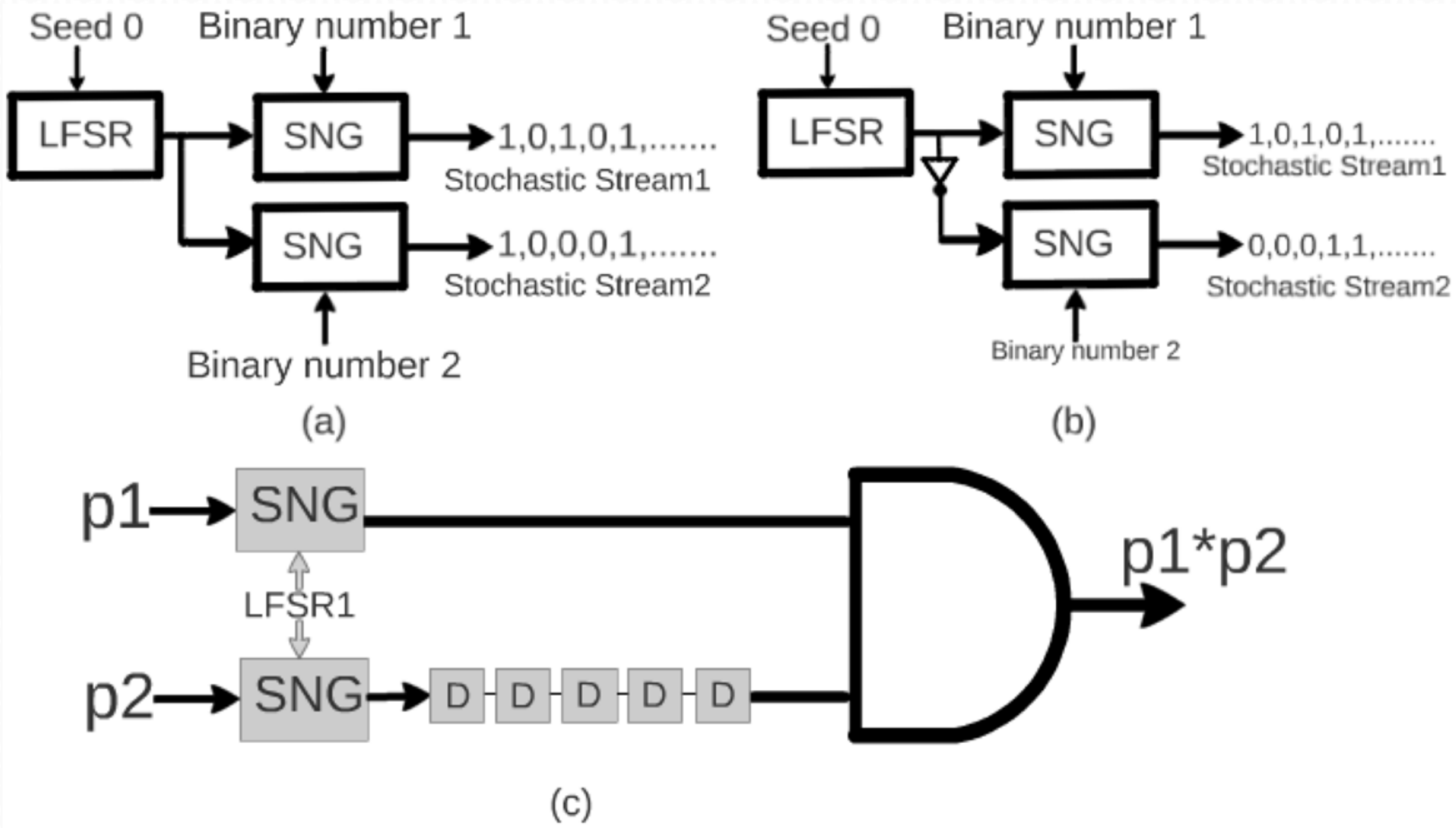
- Sum of product





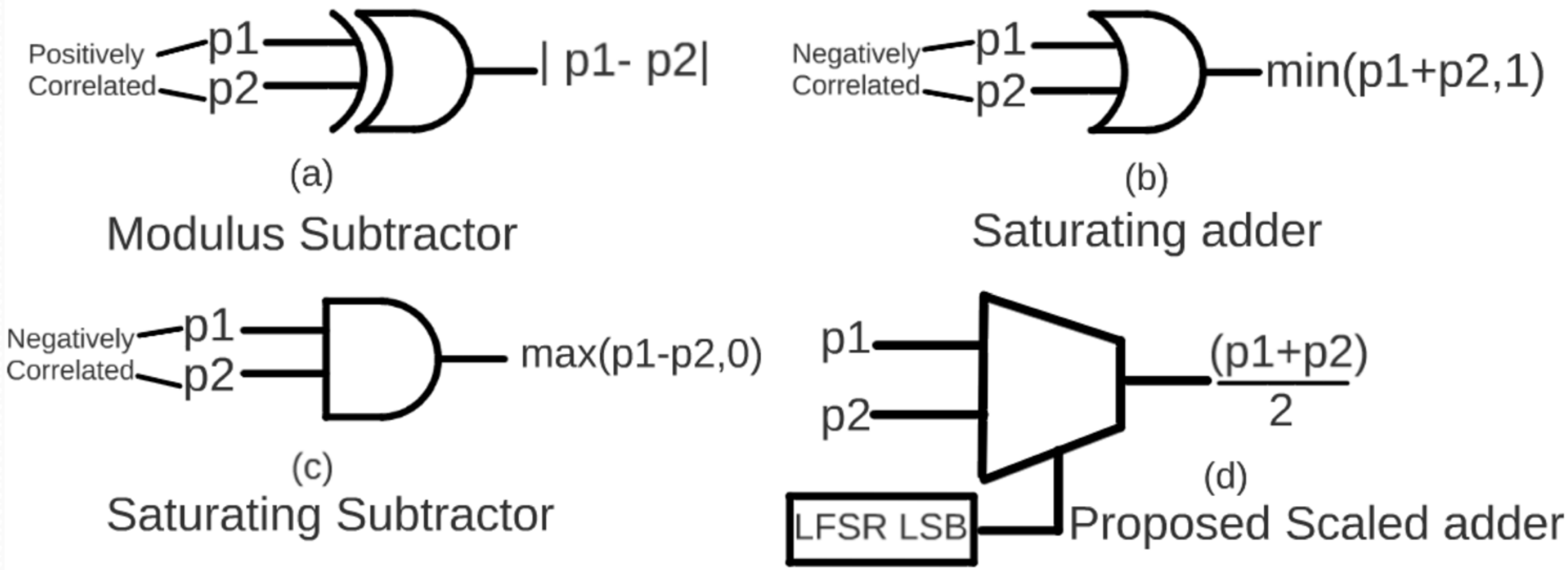
Taking Advantage of Correlation in Stochastic Computing

- Correlated inputs reduces complexity of SNG
- Correlation can be exploited wisely



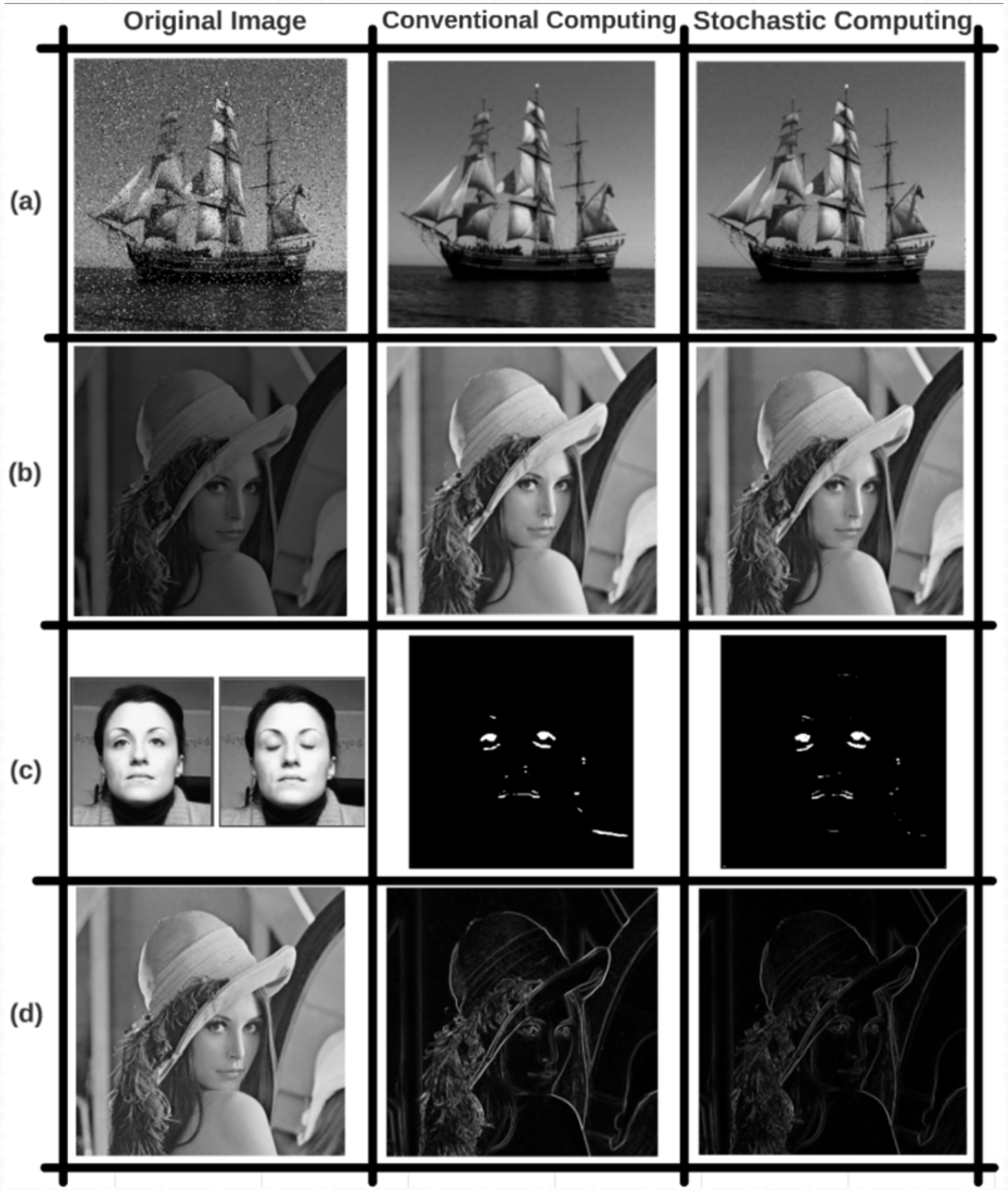
Taking Advantage of Correlation in Stochastic Computing

- Correlated inputs reduces complexity of SNG
- Correlation can be exploited wisely



Results

- Image processing
  - median filter,
  - contrast stretching
  - frame difference based image segmentation
  - edge detection
- 256-bit stochastic streams
- Implementation on Xilinx ZYNQ 706 board



R.K. Bhudhwani, R. Ragavan, O. Sentieys. Taking advantage of correlation in stochastic computing, IEEE ISCAS, 2017.

Results

- Conventional, existing, and proposed SC
- Accuracy, area, and delay
  - Mean output accuracy reduction per pixel

Benchmarks	Conventional Implementation		
	Mean Accuracy reduction per pixel (%)	Area (LUTs)	Delay (ns)
Median Filter	0.00	234	15.98
Contrast Stretching	0.00	291	24.04
Frame Segmentation	0.00	16	3.88
Edge Detection	0.00	116	4.39

Existing Stochastic Implementation			Proposed Stochastic Implementation		
Mean Accuracy reduction per pixel (%)	Area (LUTs)	Delay (ns)	Mean Accuracy reduction per pixel (%)	Area (LUTs)	Delay (ns)
1.82	478	5921.5	0.00	50	903.42
4.96	42	921.08	3.11	22	573.44
0.82	43	1062.91	0.52	21	860.16
6.8	98	2361.6	4.25	45	767.23

Soft Error Injection

- SC is more tolerant to fault injection

	Mean Accuracy reduction per pixel (%)					
	Conventional Implementation			Proposed Stochastic Implementation		
Soft Error	0%	10%	20%	0%	10%	20%
Median Filter	0.00	2.39	4.21	0.00	1.12	1.24
Contrast Stretching	0.00	10.42	18.69	3.11	6.81	9.69
Frame Segmentation	0.00	11.57	20.57	0.52	1.52	2.26
Edge Detection	0.00	8.76	18.48	4.25	5.12	7.26

R.K. Bhudhwani, R. Ragavan, O. Sentieys. Taking advantage of correlation in stochastic computing, IEEE ISCAS, 2017.

Conclusion (SC)

- SC provides massive low area, parallelism, error tolerance
- Only suitable for low-precision
- High processing latency

- Exploiting correlation
  - improves accuracy by 37% on average
  - Reduction of 50-90% in area and 20-85% in delay

Conclusions

- Most applications tolerate imprecision
- Playing with accuracy is an effective way to save energy consumption
  - Word-length
  - Number representations, including exotic ones
- Not only computation, but also memory and transfers
- Run-time accuracy adaptation would increase energy efficiency even further
- Analytical accuracy models are key to scalability of exploration techniques