Evolutionary Approximation of Software for Embedded Systems: Median Function

Vojtech Mrazek, Zdenek Vasicek, Lukas Sekanina

Faculty of Information Technology
Brno University of Technology,
Czech Republic

Outline

- Approximate computing
- Median function
  - properties, implementation, application in image processing
- Evolutionary approximation of median function
  - the proposed method
  - analysis of the results for real microcontrollers

Approximate computing

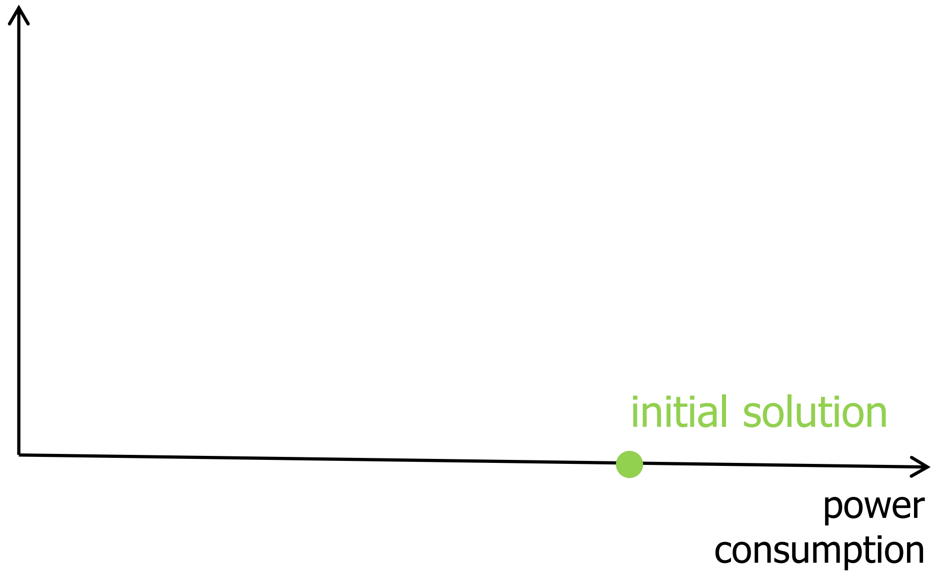
- Motivation: many real-world applications are error-resilient
- Principle: relaxation in accuracy can be used to simplify the complexity of computations and reduce the power consumption
- Applicability: 83% of runtime spent in computations can be approximated

V. K. Chippa, S. T. Chakradhar, K. Roy and A. Raghunathan, *Analysis and characterization of inherent application resilience for approximate computing*, DAC 2013.

Approximate computing

- Motivation: many real-world applications are error-resilient
- Principle: relaxation in accuracy can be used to simplify the complexity of computations and reduce the power consumption
- Applicability: 83% of runtime spent in computations can be approximated

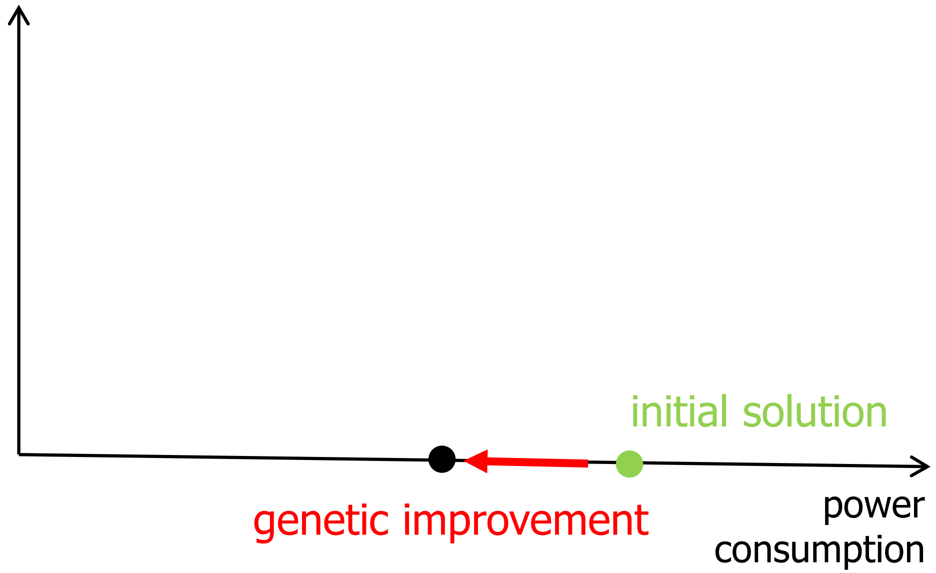
V. K. Chippa, S. T. Chakradhar, K. Roy and A. Raghunathan, *Analysis and characterization of inherent application resilience for approximate computing*, DAC 2013.



Approximate computing

- Motivation: many real-world applications are error-resilient
- Principle: relaxation in accuracy can be used to simplify the complexity of computations and reduce the power consumption
- Applicability: 83% of runtime spent in computations can be approximated

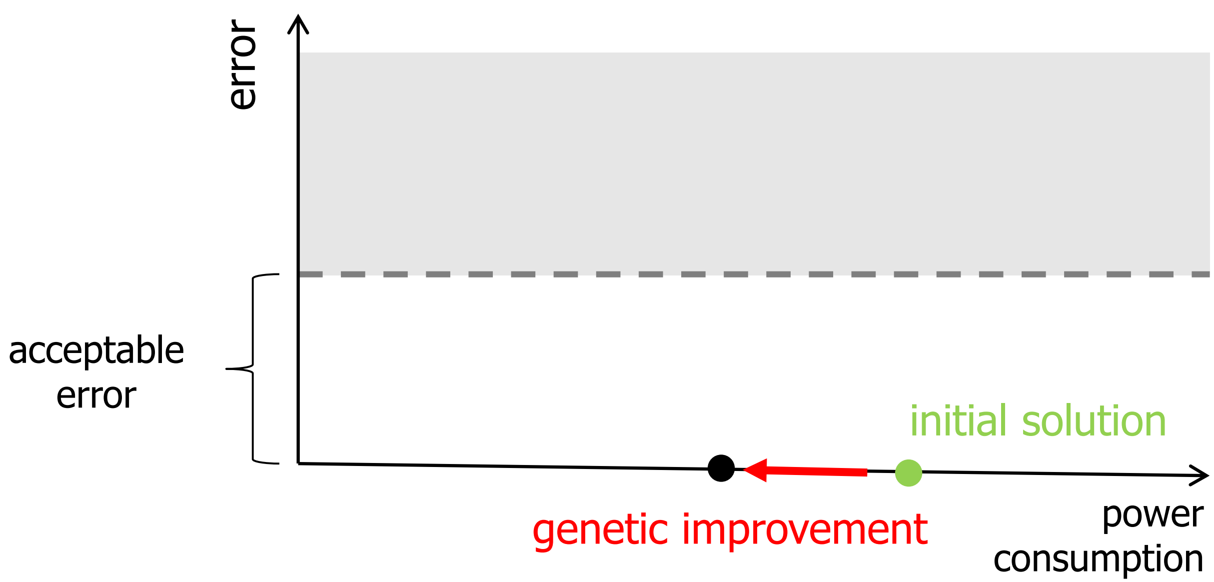
V. K. Chippa, S. T. Chakradhar, K. Roy and A. Raghunathan, *Analysis and characterization of inherent application resilience for approximate computing*, DAC 2013.



Approximate computing

- Motivation: many real-world applications are error-resilient
- Principle: relaxation in accuracy can be used to simplify the complexity of computations and reduce the power consumption
- Applicability: 83% of runtime spent in computations can be approximated

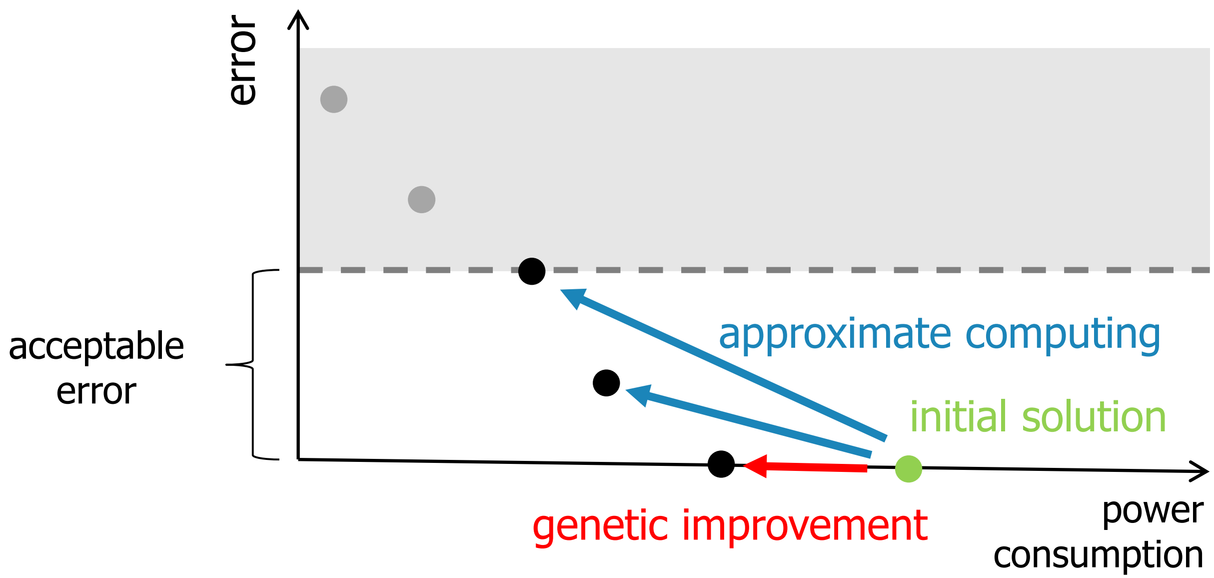
V. K. Chippa, S. T. Chakradhar, K. Roy and A. Raghunathan, *Analysis and characterization of inherent application resilience for approximate computing*, DAC 2013.



Approximate computing

- Motivation: many real-world applications are error-resilient
- Principle: relaxation in accuracy can be used to simplify the complexity of computations and reduce the power consumption
- Applicability: 83% of runtime spent in computations can be approximated

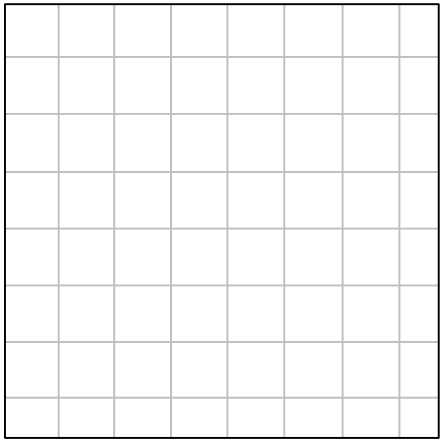
V. K. Chippa, S. T. Chakradhar, K. Roy and A. Raghunathan, *Analysis and characterization of inherent application resilience for approximate computing*, DAC 2013.



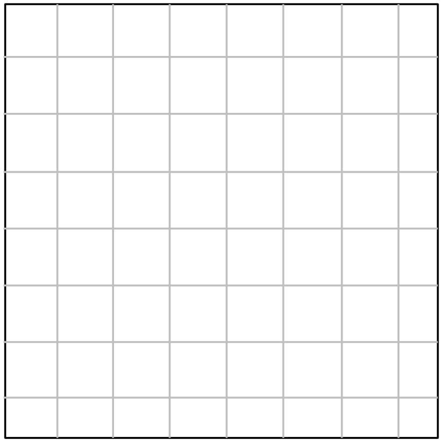
Median function

- Median: a value separating a finite sequence of data samples to two halves
- Typical application: smoothing of acquired (measured) data
- Example: noise removal in an image using a concept of sliding window

- Median: a value separating a finite sequence of data samples to two halves
- Typical application: smoothing of acquired (measured) data
- Example: noise removal in an image using a concept of sliding window



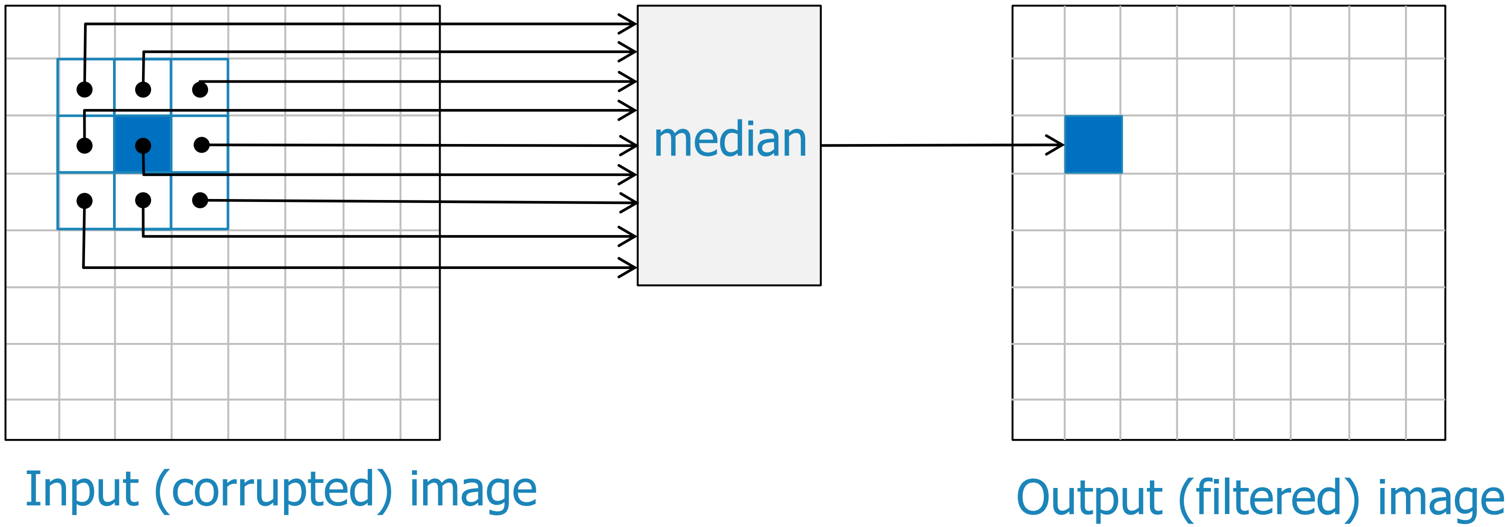
Input (corrupted) image



Output (filtered) image

Median function

- Median: a value separating a finite sequence of data samples to two halves
- Typical application: smoothing of acquired (measured) data
- Example: noise removal in an image using a concept of sliding window



Median in image processing

corrupted image
(10% pixels, impulse noise)

Median in image processing

corrupted image
(10% pixels, impulse noise)

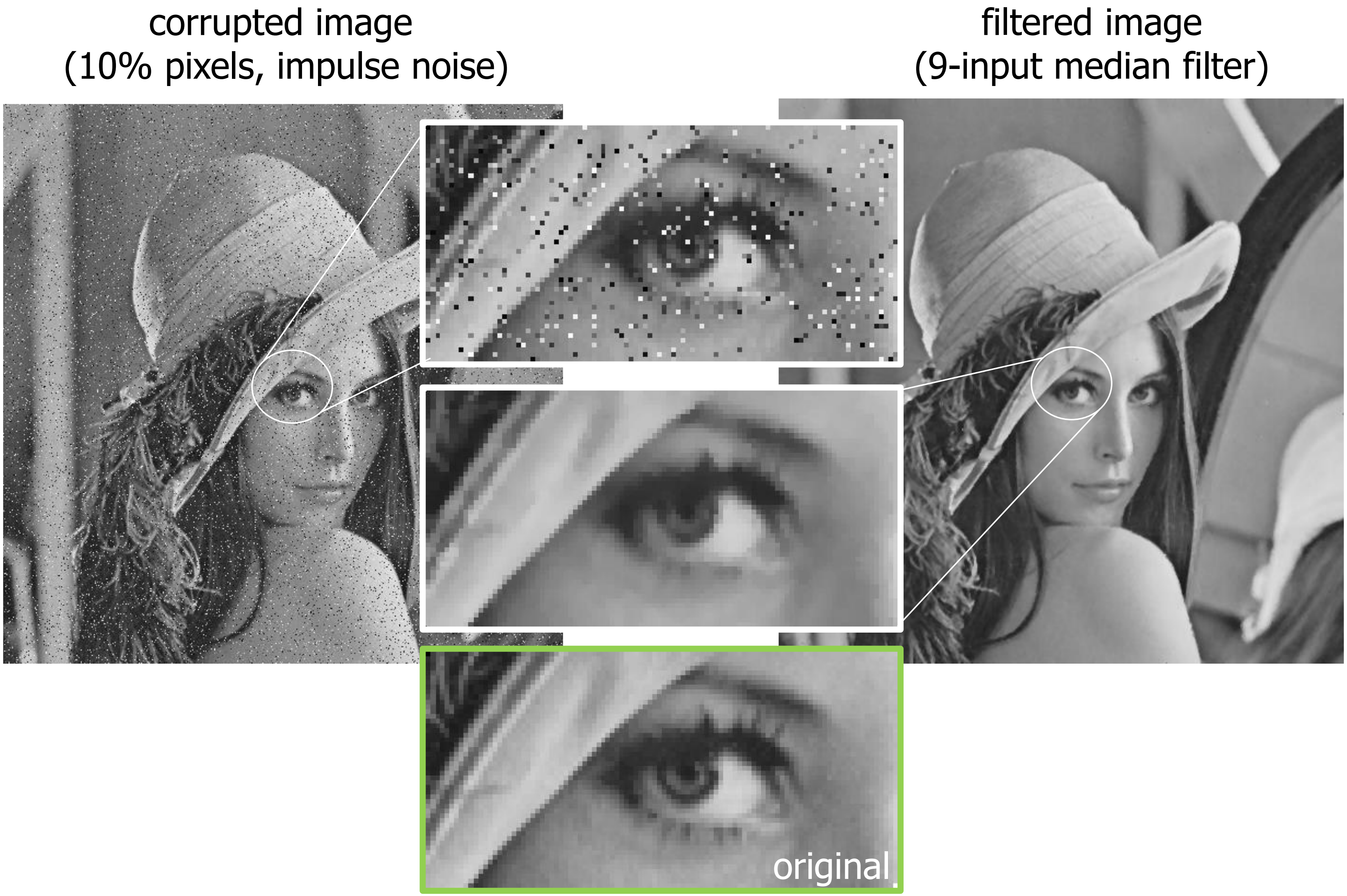
filtered image
(9-input median filter)

Median in image processing

corrupted image
(10% pixels, impulse noise)

filtered image
(9-input median filter)

Median in image processing



Implementation of median filter

- To determine the median, we can employ:
  - a sorting algorithm
  - a selection algorithm
  - a median network

- Median network
  - a structure consisting of compare & swap operations
  - an optimal network is known for some sizes

```
#define PIX_SORT(a,b) {
 if ((a)>(b))
   PIX_SWAP((a),(b));
}
```

```
pixelvalue opt_med9 (pixelvalue * p)
{
    PIX_SORT(p[1], p[2]) ; PIX_SORT(p[4], p[5]) ; PIX_SORT(p[7], p[8]) ;
    PIX_SORT(p[0], p[1]) ; PIX_SORT(p[3], p[4]) ; PIX_SORT(p[6], p[7]) ;
    PIX_SORT(p[1], p[2]) ; PIX_SORT(p[4], p[5]) ; PIX_SORT(p[7], p[8]) ;
    PIX_SORT(p[0], p[3]) ; PIX_SORT(p[5], p[8]) ; PIX_SORT(p[4], p[7]) ;
    PIX_SORT(p[3], p[6]) ; PIX_SORT(p[1], p[4]) ; PIX_SORT(p[2], p[5]) ;
    PIX_SORT(p[4], p[7]) ; PIX_SORT(p[4], p[2]) ; PIX_SORT(p[6], p[4]) ;
    PIX_SORT(p[4], p[2]) ; return(p[4]) ;
}
```

Source: http://ndevilla.free.fr/median/median.pdf

Implementation of median filter

- Alternatively, max and min operations can be used
  - the sequence of operations is invariant w.r.t. the input data
  - suitable for HW architectures equipped with MIN/MAX instruction
  - easier evaluation of the correctess (zero-one theorem, AND/OR)

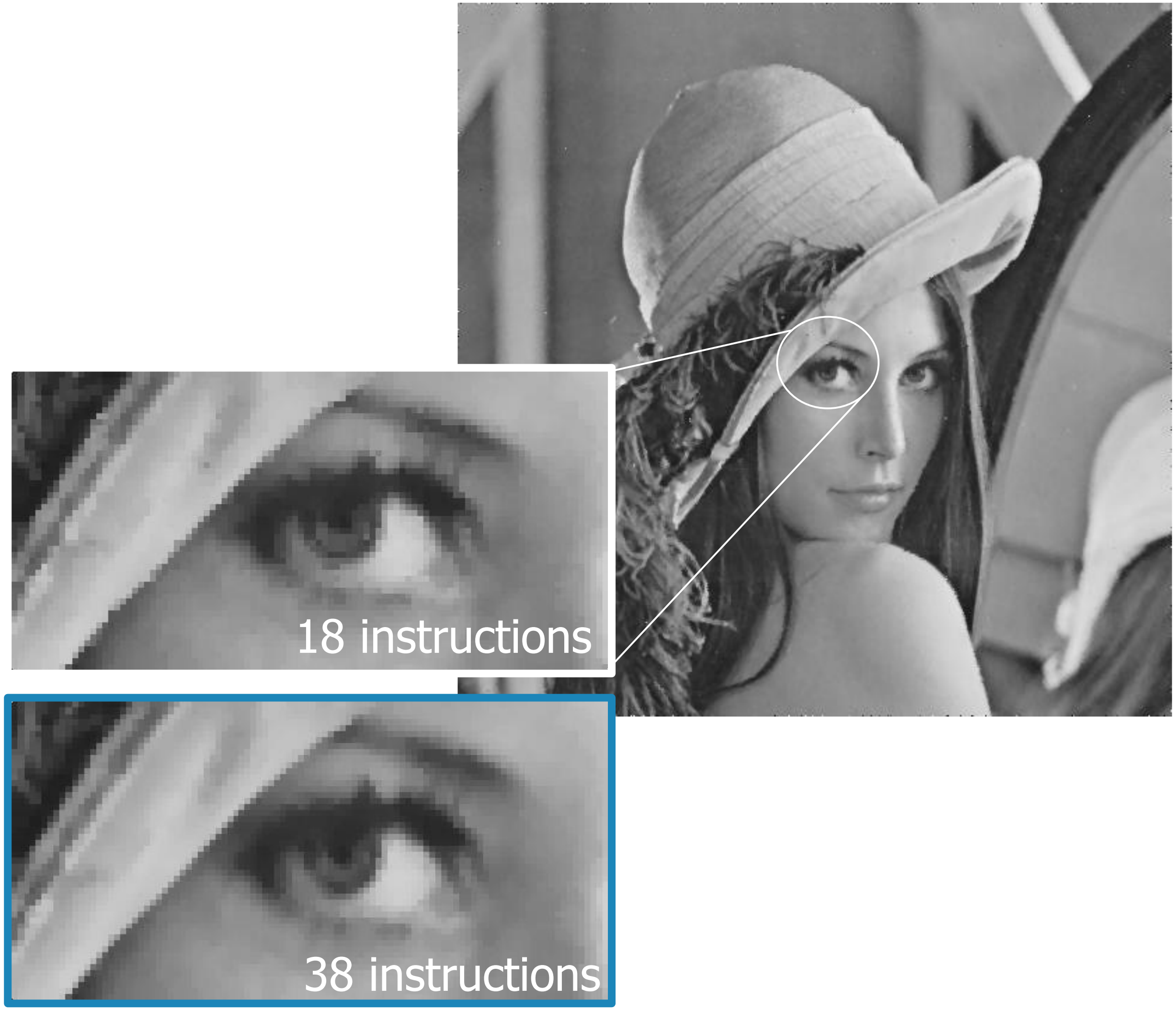
```
pixelvalue  approx_med9 (pixelvalue * p)
{
          pixelvalue  s00=MIN(p[2],p[3]), s01=MAX(p[5],p[4]), s02=MAX(p[2],p[3]);
          pixelvalue  s03=MIN(p[4],p[5]), s04=MIN(p[0],p[1]), s05=MAX(p[7],p[6]);
          pixelvalue  s06=MIN(p[8],s05) , s07=MAX(p[0],p[1]), s08=MAX(s04,s00)  ;
          pixelvalue  s09=MAX(s08,s03)  , s10=MIN(p[6],p[7]), s12=MIN(s01,s07)  ;
          pixelvalue  s13=MIN(s12,s02)  , s14=MAX(s06,s09)  , s15=MIN(s06,s09)  ;
          pixelvalue  s16=MAX(s13,s15)  , s17=MAX(s10,s16)  , s18=MIN(s14,s17)  ;

          return s18;
}
```

Approximate median – 18 operations

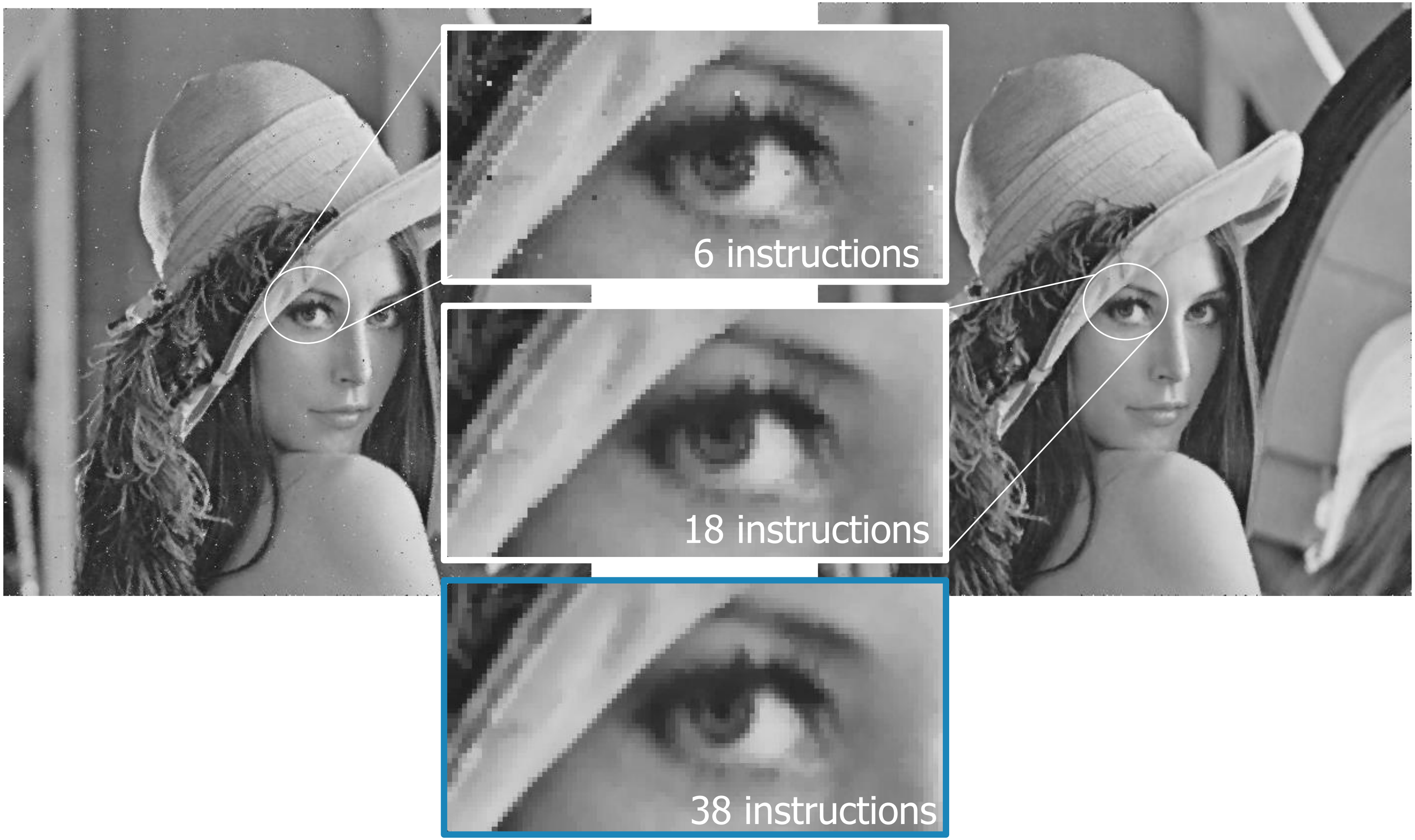
Approximate median filter

filtered image
(9-input median filter – 18 instructions)



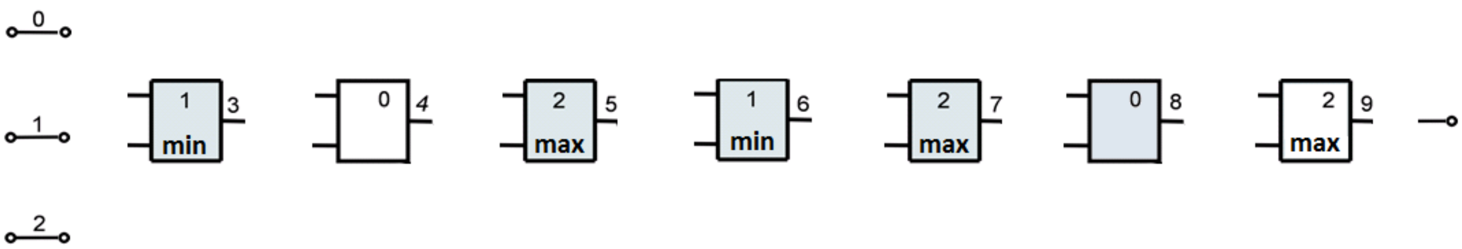
filtered image
(9-input median filter – 6 instructions)

filtered image
(9-input median filter – 18 instructions)



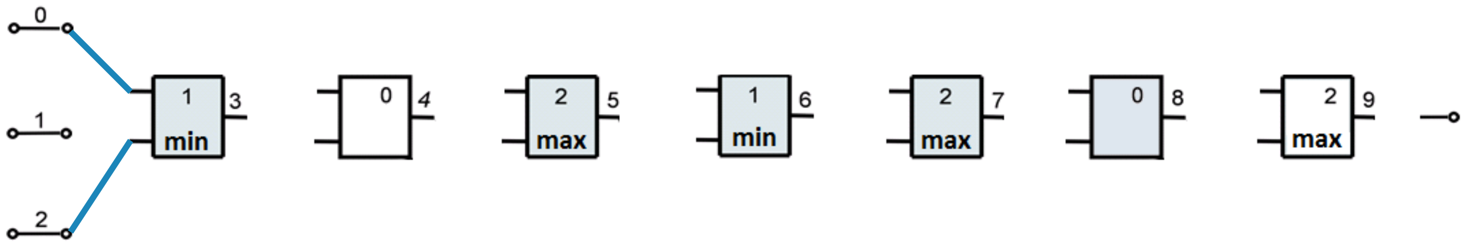
Approximate circuit design by means of CGP

- Median network (consisting of up to N operations) is represented by means of an one-dimensional array of N nodes.
- Each node can act as: identity (0), minimum (1), maximum (2)
- Each node can be connected to a node situated in the previous columns (no feedbacks are allowed).
- The configuration of nodes (the function and connection) is encoded using $3N+1$ integers.



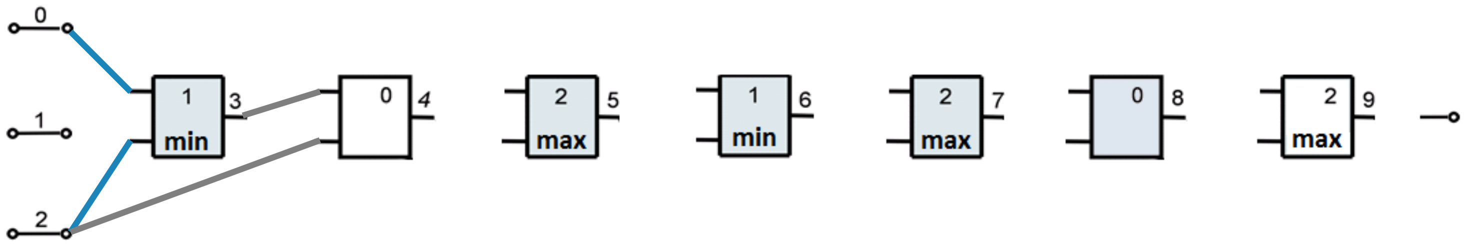
Chromosome: 0, 2, <u>3</u>; 3, 2, <u>0</u>; 0, 2, <u>2</u>; 5, 3, <u>1</u>; 6, 1, <u>2</u>; 7, 0, <u>0</u>; 6, 8, <u>2</u>; **8**

- Median network (consisting of up to N operations) is represented by means of an one-dimensional array of N nodes.
- Each node can act as: identity (0), minimum (1), maximum (2)
- Each node can be connected to a node situated in the previous columns (no feedbacks are allowed).
- The configuration of nodes (the function and connection) is encoded using $3N + 1$ integers.



Chromosome: 0, 2, 3; 3, 2, 0; 0, 2, 2; 5, 3, 1; 6, 1, 2; 7, 0, 0; 6, 8, 2; **8**

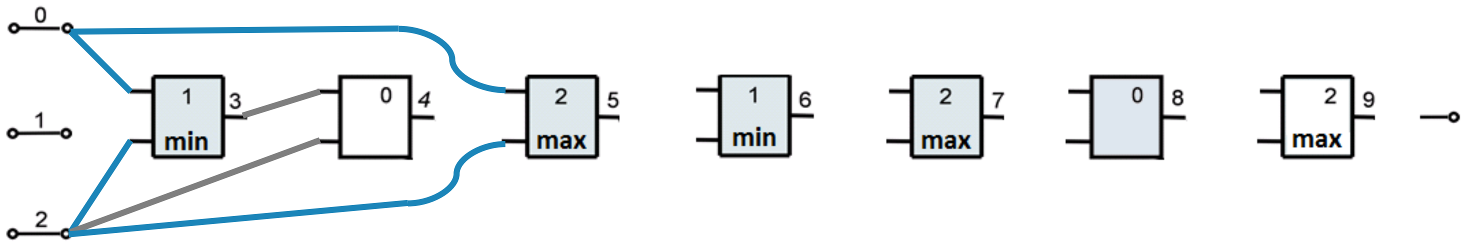
- Median network (consisting of up to N operations) is represented by means of an one-dimensional array of N nodes.
- Each node can act as: identity (0), minimum (1), maximum (2)
- Each node can be connected to a node situated in the previous columns (no feedbacks are allowed).
- The configuration of nodes (the function and connection) is encoded using $3N + 1$ integers.



Chromosome: 0, 2, 3; 3, 2, 0; 0, 2, 2; 5, 3, 1; 6, 1, 2; 7, 0, 0; 6, 8, 2; **8**

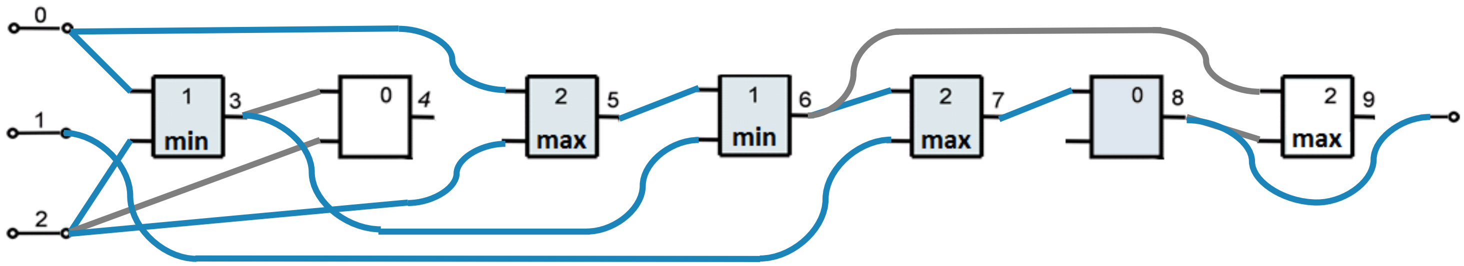
Approximate circuit design by means of CGP

- Median network (consisting of up to N operations) is represented by means of an one-dimensional array of N nodes.
- Each node can act as: identity (0), minimum (1), maximum (2)
- Each node can be connected to a node situated in the previous columns (no feedbacks are allowed).
- The configuration of nodes (the function and connection) is encoded using $3N + 1$ integers.



Chromosome: 0, 2, 3; 3, 2, 0; 0, 2, 2; 5, 3, 1; 6, 1, 2; 7, 0, 0; 6, 8, 2; **8**

- Median network (consisting of up to N operations) is represented by means of an one-dimensional array of N nodes.
- Each node can act as: identity (0), minimum (1), maximum (2)
- Each node can be connected to a node situated in the previous columns (no feedbacks are allowed).
- The configuration of nodes (the function and connection) is encoded using $3N+1$ integers.



Chromosome: 0, 2, 3; 3, 2, 0; 0, 2, 2; 5, 3, 1; 6, 1, 2; 7, 0, 0; 6, 8, 2; **8**

The fitness function

- The quality of approximation is measured as the sum of absolute differences between the output value of a candidate solution and reference

$$error = \sum_{i \in S} \left| O_{candidate}(i) - O_{reference}(i) \right|$$

- Scalability issue
  - |S| could be reduced from 2^{8^n} to 2^n using the zero-one principle.
  - However, it would be impossible to reasonably quantify the error (It is not important, how many invalid responses are produced).

- Solution
  - Use a randomly generated subset of S of a “reasonable” size

Z. Vasicek and L. Sekanina. *Evolutionary approach to approximate digital circuits design*. IEEE trans. on Evolutionary Computation, Vol. 19 , No. 3, 2015

Evolutionary design of approximate medians

- Resource-oriented design approach is employed.
  - The evolutionary approximation exploits the idea that CGP is capable of minimizing the error even if the number of available functional nodes is not sufficient for obtaining a fully functional solution.
- Experimental setup:
  - (1+4)-ES, no crossover, 5 % of the chromosome mutated

	Median-9	Median-25
Inputs	9	25
Outputs	1	1
Generations	3×10^6 (3 hours)	3×10^5 (3 hours)
Training vectors	1×10^4	1×10^5
Reference solution	38 operations	220 operations
Number of nodes	6 – 34 operations	10 – 200 operations

- The principle of construction of a median network guarantees that the output value is always one of the input values.

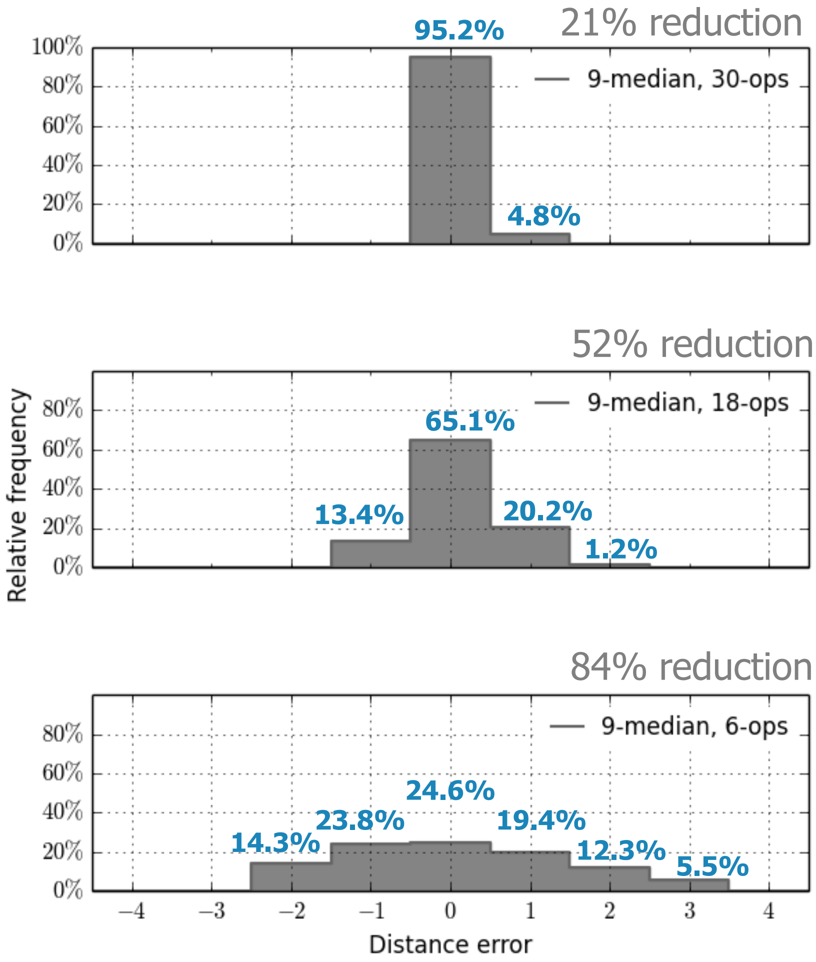
- Consequence:
  - If a sequence of $2n + 1$ successive numbers $\mathrm{R} = [-n, \ldots, n]$ is used as the input, the absolute value of the highest obtained number equals to the worst-case distance from $(n+1)^{\text{th}}$ lowest element

$$\text{median}_{(2n+1)}(\{-n, -n+1, \ldots, 0, \ldots, n-1, n\}) = 0$$

  - Permutations of R can be used instead of all possible input combinations
    - 9-median: 3.62×10^{5} permutations (vs. 6.27×10^{21} combinations)
    - 25-median: 1.25×10^{25} permutations (vs. 4.20×10^{60} combinations)

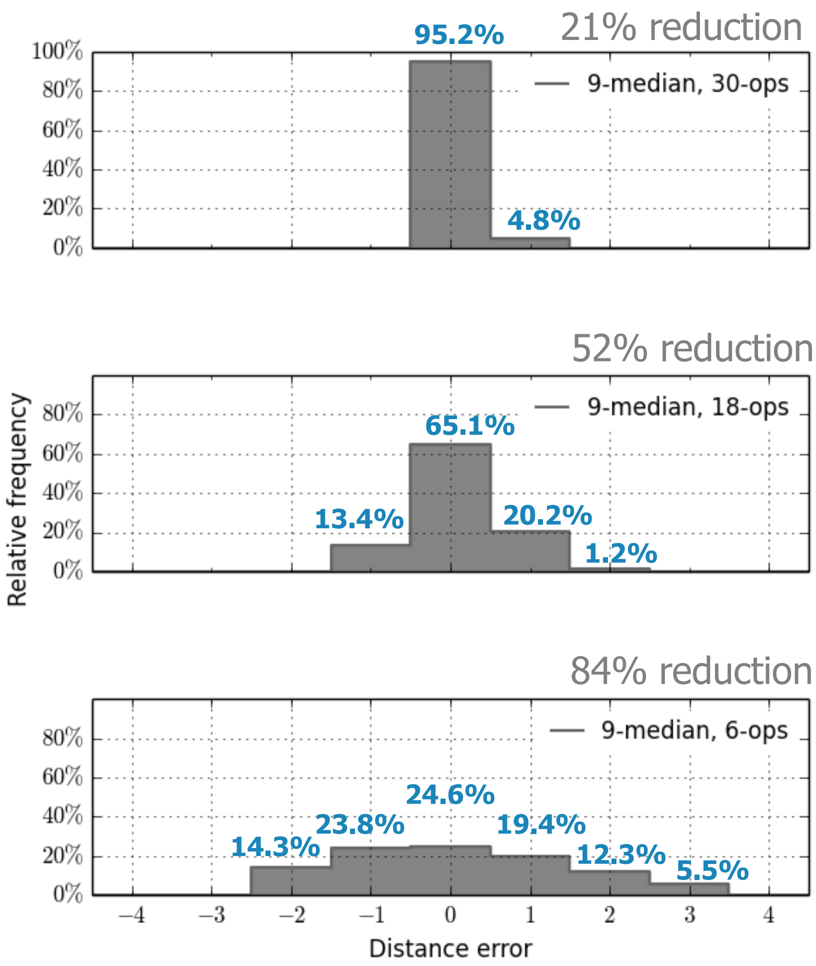
9-input median

fully-working: 38 operations



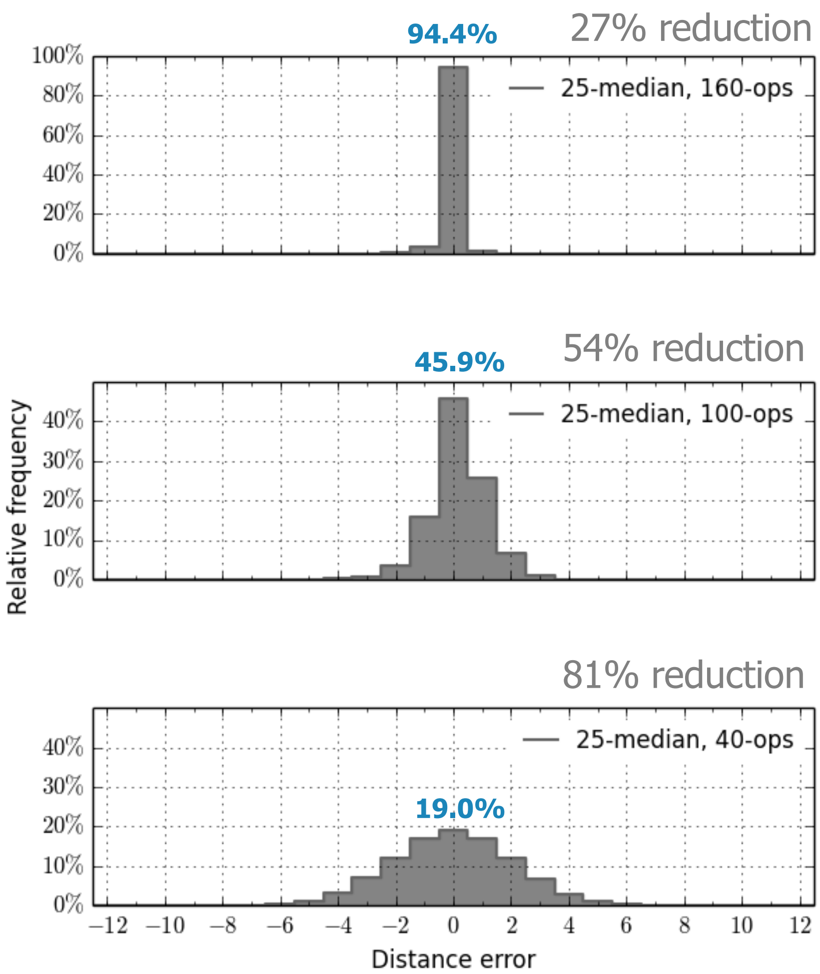
9-input median

fully-working: 38 operations



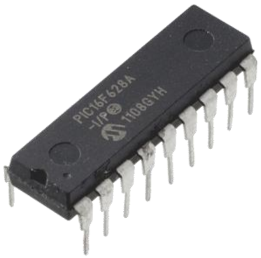
25-input median

fully-working: 220 operations



Target platforms:

- Microchip PIC16F628
  - 8 bit microprocessor
  - accumulator architecture



- Microchip PIC24F08
  - 16 bit microprocessor
  - register architecture

- ST STM32F100RB
  - 32 bit microprocessor
  - ARM Cortex M3 core

Power consumption measured on real chips.

Impl.	Time [μs]			Energy [nWs]		
	STM32	PIC24	PIC16	STM32	PIC24	PIC16
6-ops	2.8	54.5	170.5	86	377	342
10-ops	3.3	70.8	251.5	102	490	504
14-ops	3.9	86.8	336.5	118	600	674
18-ops	4.5	104.5	424.1	138	723	850
22-ops	5.0	116.7	487.8	151	808	978
26-ops	5.9	130.0	558.0	179	900	1118
30-ops	6.0	142.0	627.4	181	983	1257
34-ops	6.4	154.0	819.7	196	1066	1643
38-ops	6.9	165.5	885.0	210	1145	1774
qsort	28.5	1106.2	—	869	7655	—

Execution time and power consumption 9-median

Impl.	Time [μs]			Energy [nWs]		
	STM32	PIC24	PIC16	STM32	PIC24	PIC16
6-ops	2.8	54.5	170.5	86	377	342
10-ops	3.3	70.8	251.5	102	490	504
14-ops	3.9	86.8	336.5	118	600	674
18-ops	4.5	104.5	424.1	138	723	850
22-ops	5.0	116.7	487.8	151	808	978
26-ops	5.9	130.0	558.0	179	900	1118
30-ops	6.0	142.0	627.4	181	983	1257
34-ops	6.4	154.0	819.7	196	1066	1643
38-ops	6.9	165.5	885.0	210	1145	1774
qsort	28.5	1106.2	—	869	7655	—

fully-working median

- Quick-sort based implementation is slower and consumes significantly more energy compared to the median network.
- Due to the limited resources, quick-sort can’t be even implemented on PIC16.

Impl.	Time [μs]			Energy [nWs]		
	STM32	PIC24	PIC16	STM32	PIC24	PIC16
6-ops	2.8	54.5	170.5	86	377	342
10-ops	3.3	70.8	251.5	102	490	504
14-ops	3.9	86.8	336.5	118	600	674
18-ops	4.5	104.5	424.1	138	723	850
22-ops	5.0	116.7	487.8	151	808	978
26-ops	5.9	130.0	558.0	179	900	1118
30-ops	6.0	142.0	627.4	181	983	1257
34-ops	6.4	154.0	819.7	196	1066	1643
38-ops	6.9	165.5	885.0	210	1145	1774
qsort	28.5	1106.2	—	869	7655	—

4.8% error prob.,
max. error dist. 1
21% power reduction

fully-working median

- Quick-sort based implementation is slower and consumes significantly more energy compared to the median network.
- Due to the limited resources, quick-sort can't be even implemented on PIC16.
- 21% reduction in power consumption was achieved in the case of 30-ops median providing a negligible error

Impl.	Time [μs]			Energy [nWs]		
	STM32	PIC24	PIC16	STM32	PIC24	PIC16
6-ops	2.8	54.5	170.5	86	377	342
10-ops	3.3	70.8	251.5	102	490	504
14-ops	3.9	86.8	336.5	118	600	674
18-ops	4.5	104.5	424.1	138	723	850
22-ops	5.0	116.7	487.8	151	808	978
26-ops	5.9	130.0	558.0	179	900	1118
30-ops	6.0	142.0	627.4	181	983	1257
34-ops	6.4	154.0	819.7	196	1066	1643
38-ops	6.9	165.5	885.0	210	1145	1774
qsort	28.5	1106.2	—	869	7655	—

34.9% error prob.,
max. error dist. 2
52% power reduction

4.8% error prob.,
max. error dist. 1
21% power reduction

fully-working median

- Quick-sort based implementation is slower and consumes significantly more energy compared to the median network.
- Due to the limited resources, quick-sort can’t be even implemented on PIC16.
- 21% reduction in power consumption was achieved in the case of 30-ops median providing a negligible error

Impl.	Time [μs]		Energy [nWs]	
	STM32	PIC24	STM32	PIC24
10-ops	3.4	71.5	104	495
40-ops	8.1	188.5	246	1304
70-ops	13.3	303.0	406	2097
100-ops	17.3	401.6	528	2779
130-ops	22.1	491.2	673	3399
160-ops	27.4	581.4	836	4023
170-ops	29.1	609.8	888	4220
200-ops	34.8	698.3	1063	4832
220-ops	39.3	755.3	1200	5227
qsort	101.6	3067.5	3099	21227

- 25-input median consisting of up to 220 operations offers a higher potential for power savings.
- There is nearly linear dependency between the number of operations and consumed energy (approx. 5 nW per operation for STM32).
- PIC24 requires five times more energy to accomplish the same operation.

Conclusions

- A new approach to the approximation of software routines for MCUs was presented.
- We confirmed that CGP is able to find a good trade off between error and code size even if the code size is intentionally constrained.
- A significant improvement in power consumption, code size and time of execution was achieved.
- A new method for analysis of quality of the proposed approximations was proposed.