Floating-point Precision vs Speed

June 25, 2020

1 Introduction to Floating-point system

Floating points are one of the few ways in which we can represent real numbers in computer systems. Floating-point systems consist of three parts.

1. Sign - Indicates if the number is positive or negative. This part takes always 1 bit, irrespective system precision.
2. Exponent - Natural number belonging to the range [0.2 * e-1]. e is the number of bits we put into the exponent. In the end, we shift this range into left by bias to be able to represent negative exponents.
3. Mantissa - Normalized mantissa, fractional number in the range [1, B). Where B is the so-called base. For computer systems B = 2.

Let's look at the above parts for the 32 bit IEEE-754 single format.

b_{31}	b_{30}.....$b_{24}b_{23}$	b_{22}.....$b_3b_2b_1b_0$
Sign	Exponent	Mantissa

Figure 1: IEEE-754 single floating-point system

To get a floating point number in the IEEE-754 single standard, we'll use the following transformations

$$exponent = \sum_{n=0}^{7} b_{(23+n)} \times 2^n \tag{1}$$

$$mantisa = \sum_{n=0}^{22} b_{(22-n)} \times 2^{-n} \tag{2}$$

Using the above transformations, we can write any floating point number in the form.

$$(-1)^s \times mantisa \times 2^{(exponent-127)} \tag{3}$$

2 Properties of floating-point numbers

2.1 Floating-point accuracy

In every floating-point format there is a machine epsilon ϵ_m, that tells us how good our calculation precision is. The lower the value of machine epsilon, the greater the precision of calculations. The above value is directly related to the number of bits devoted to the mantissa. If we devote b_n bits to the mantissa, then the value of the machine epsilon will be:

$$\epsilon_m = 2^{-b_n} \tag{4}$$

For IEEE-754 single:

$$\epsilon_m = 2^{-23} \approx 1.19 \times 10^{-7} \tag{5}$$

Which means we have an accuracy of 7 decimal places. The number of bits allocated to the mantissa is closely related to precision. And what is the value of the exponent related to? It tells us how big and how small, but not negative numbers we can write in our system.

2.2 The smallest and the largest value

We can use transformations (1) and (2) to determine the smallest and the largest value for given floating-point format. Let's calculate those values once more for IEEE-754 single standard. Figure 2 sums up all properties.

$$f_{max+} = 2^{exponent_{max}-127} \times (1 - 2^{-b_n}) \tag{6}$$

$$f_{min+} = 2^{exponent_{min}-127} \tag{7}$$

IEEE-754 single		
f_{min}	f_{max}	machine epsilon
0.293×10^{-38}	3.4×10^{38}	$2^{-23} \approx 1.19 \times 10^{-7}$

Figure 2: Sum up for IEEE-754 single

But what if I would like to create my own custom floating point? How all those values would change? Let's create general transformations for every floating-point format.

$$bias = \lfloor exponent_{max}/2 \rfloor \tag{8}$$

$$f_{max+} = 2^{exponent_{max}-bias} \times (1 - 2^{-b_n}) \tag{9}$$

$$f_{min+} = 2^{exponent_{min}-bias} \tag{10}$$

Format	f_{min}	f_{max}	machine epsilon
64-bit: Double	2.22×10^{-308}	1.79×10^{308}	$2^{-53} \approx 1.11 \times 10^{-16}$
32-bit: Single	0.293×10^{-38}	3.4×10^{38}	$2^{-23} \approx 1.19 \times 10^{-7}$
16-bit: Half	6.10×10^{-5}	0.65×10^{5}	$2^{-10} \approx 0.97 \times 10^{-4}$
13-bit: Soft-ieee754	0.015625	255	$2^{-8} \approx 3.94 \times 10^{-3}$
9-bit: Soft-ieee754	0.015625	240	$2^{-4} \approx 6.25 \times 10^{-2}$
8-bit: Soft-ieee754	0.25	15	$2^{-4} \approx 6.25 \times 10^{-2}$

Figure 3: Properties of floating-point format depending on the format selected

3 Profit of lowering precision

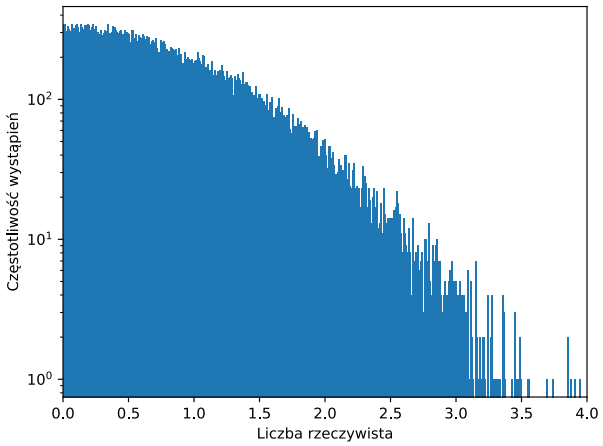
Reducing the precision of floating point numbers, reduces the requirements for the amount of memory and computing resources. Thanks to this operation, we will also speed up basic arithmetic operations such as multiplication and addition. I checked how the consumption of basic logical resources, such as flip-flops, lookup tables, and the number of clock cycles needed to perform operations for few custom and general types of floating-point changes.

Format	flip-flops	Lookup tables	clock cycles
64-bit: Double	545	2677	3
32-bit: Single	272	714	3
16-bit: Half	82	208	1
13-bit: Soft-ieee754	0	236	1
9-bit: Soft-ieee754	0	158	1
8-bit: Soft-ieee754	0	148	1

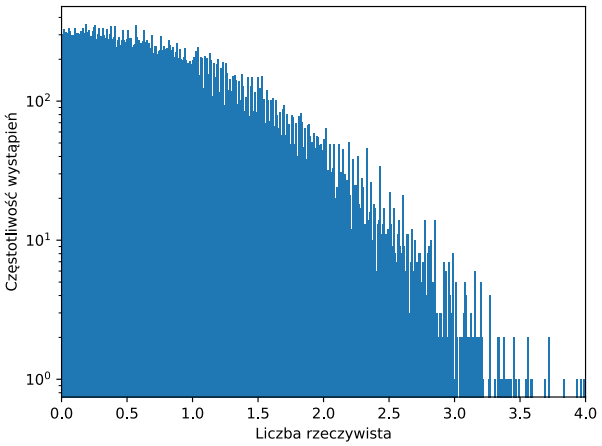
Figure 4: Consumption of basic logical resources

4 Disadvantage of lowering precision

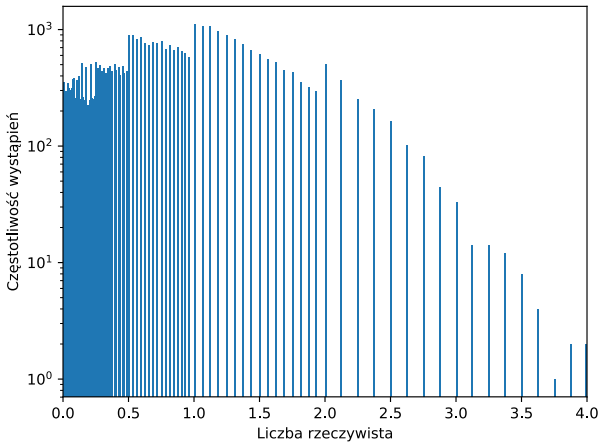
By reducing the number of bits devoted to the mantissa or the exponent, we gain memory and make arithmetic operations faster. Graphs (Figure 5) illustrating how the distribution/density of numbers changes for given floating-point number formats. 40,000 samples were generated from the normal distribution of values expected value 0 and standard deviation equal 1. Only positive samples were considered. We can observe the charts negative impact of quantization on the representativeness of floating-point numbers. Particularly visible when quantizing up to 8 bits.



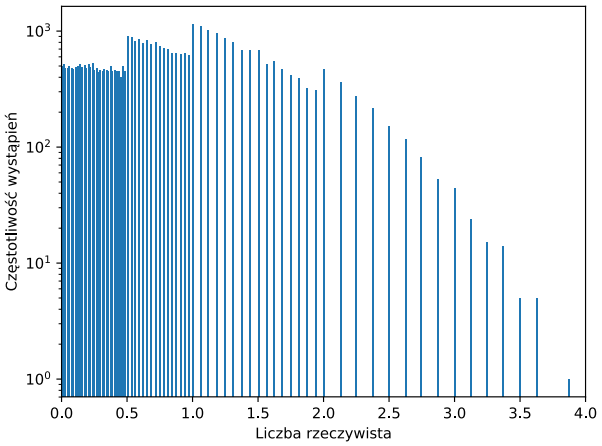
(a) IEEE-754 single format



(b) 13-bit soft-ieee754 format



(c) 9-bit soft-ieee754 format



(d) 8-bit soft-ieee754 format

Figure 5: Changes in distribution depending on the floating-point format

5 Summary

In this document I discussed pros and cons of reducing precision for floating-point and how changes in precision can affect speed/memory usage/representativeness.

References

[1] Soft-ieee754
`https://github.com/LiraNuna/soft-ieee754`

[2] Dogan Ibrahim *"SD Card Projects Using the PIC Microcontroller"*.
Chapter 1, Section 22: Floating Point Numbers. 2010.

[3] Rajaraman, V. *"EEE standard for floating point numbers"*.
https://doi.org/10.1007/s12045-016-0292-x, Reson 21, 11–30 (2016).

[4] Nicolas Limare. *"Integer and Floating-Point Arithmetic Speed vs Precision"*.
`http://nicolas.limare.net/pro/notes/2014/12/12_arit_speed/`