On the Associativity of Floating-Point Arithmetic

Sarp, Atlas and Rishikesh

Imperial College London Mathematics School

Introduction

Computers play an essential role in our daily life, performing calculations at rates that exceed natural human capability. However, these extraordinary computers have a number of limitations. One of these limitations is the inability to be able to represent certain numbers with complete accuracy. A simple number like 0.1 cannot be represented with complete accuracy within a computer, which highlights the fact that fast calculations do not mean accurate calculations. The reason behind this limitation is that computers operate on the binary number system, using only a finite number of binary digits to represent any number. As each binary digit can only have a place value which is a power of 2, it is not possible to represent every single decimal number using the binary number system. This means that the computer has to approximate the values of these decimal numbers while performing mathematical operations on them, which can lead to certain unexpected errors in the results of even simple arithmetic calculations. In our project, we have focused on a very specific arithmetic calculation and have found an expression to determine the probability that this calculation disobeys the law of associativity of addition.

Working in Binary

1 The Binary Number System

The binary system is a base-2 number system whose digits are called bits and can either be a 0 or a 1. The place value of each digit in a binary number is hence a power of 2. This is similar to how the place value of each digit in a decimal number is a power of 10.

2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1024	512	256	128	64	32	16	8	4	2	1
				1	0	1	1	0	0	1

$1011001_2 = 1 \times 64 + 1 \times 16 + 1 \times 8 + 1 \times 1 = 89$

Figure 1: An example of converting the binary number 1011001 into its decimal counterpart (1). During the conversion process, the place values of the 1s are added to the running decimal total while the place values of the 0s are not.

2 Binary Shifts

Binary shifts (or logical shifts) are operations that move all the bits of a binary number either left or right.

A left shift represents a multiplication by a factor of 2 while a right shift represents a division by a factor of 2. In this project, the symbol $\gg$ used represents a binary right shift.

Performing n number of binary shifts on a binary number produces n number of empty bit places in the resulting binary number. Hence, the resultant empty bits are replaced with n number of zeros.

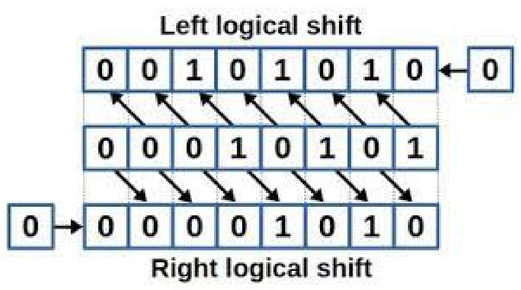

Figure 2: A visualisation depicting a left and right binary (logical) shift (2). The zeros inserted into the resulting binary number are the zeros used to fill the empty bit places.

Rounding Towards Zero

Rounding towards zero in floating-point arithmetic refers to truncating a floating-point result to produce an integer result. This is one of the methods of handling overflow and underflow in a computer, preventing it from crashing, but this will generate an error in the result. In this project, the symbols $\widehat{+}$ and $\widehat{-}$ are used to represent round-towards-zero addition and round-towards-zero subtraction respectively.

Some examples of round-towards-zero addition and round-towards-zero subtraction are as follows:

$$8.9 \mathbin{\widehat{+}} 6.7 = 15$$
$$8.7 \mathbin{\widehat{-}} 6.9 = 1.$$

Underflow

Every binary number in a computer is allocated a specific bit-length. Therefore, performing a binary shift will truncate any bits that have been shifted beyond the highest or the lowest operable bit position. This can be seen in Figure 2 where the leading 0 has been truncated after the left shift and the trailing 1 has been truncated after the right shift.

Considering the 4-bit long binary number 1111 (155 in decimal), a right shift would produce a 5-bit long binary number 0111.1 which contains a binary point. The last bit of this binary number (which is a 1) is now beyond the lowest operable bit position, and produces a situation called underflow. In order to handle the underflow, the computer will truncate any bits that are beyond the lowest operable bit position. This would result in the binary number 0111 (7 in decimal). So, the right shift produces a small inaccuracy in the result for the half of 15. This inaccuracy is what causes the unexpected errors in the results of arithmetic calculations.

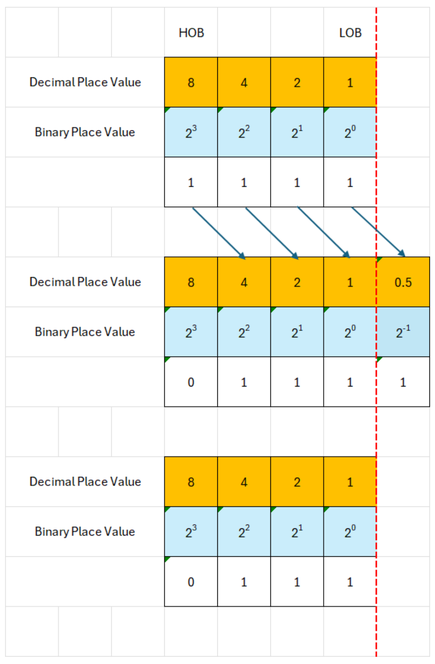

Figure 3: A visualisation depicting a right binary shift by one bit. This right binary shift produces an underflow, which has been handled by truncating the last bit which is now beyond the lowest operable bit position. Key: MOB refers to 'Most Operable Bit' and LOB refers to 'Least Operable Bit'.

The Floating-Point Problem

The floating-point representation of numbers is used to represent real numbers in a way that supports a wide range of values, including large numbers and fractional values, while using a relatively smaller number of bits compared to the standard binary format. The floating-point format for a positive real number is as follows:

$$1.mantissa \times 2^{exponent}.$$

The floating-point representation of numbers can also be colloquially referred to as 'Binary Standard Form'. Our report delves deeper into how the floating-point representation of numbers can be understood.

In this project, we have focused on finding an expression for the probability that the following floating-point inequality is true:

$$(C_F \mathbin{\widehat{+}} B_F) \mathbin{\widehat{-}} A_F \neq C_F \mathbin{\widehat{+}} (B_F \mathbin{\widehat{-}} A_F).$$

This is a rather complex floating-point problem, which we have decomposed into six simplified integer problems. The floating-point numbers A_F, B_F and C_F have been converted into the integers A, B and C by replacing the idea of exponents in the floating-point numbers with the variables m, h and k.

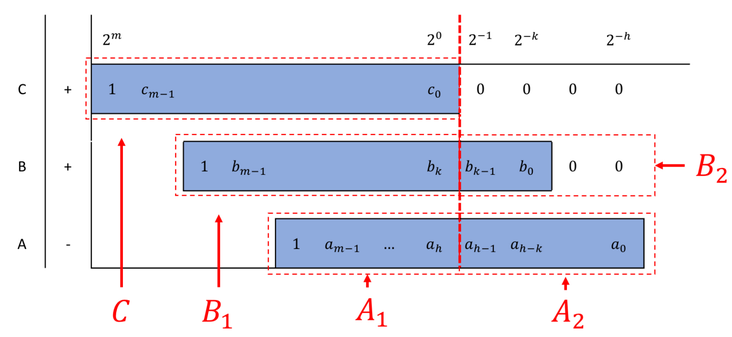

Figure 4: A visualisation depicting the integers A, B and C after the integers A and B have been right shifted by h and k bits respectively.

Simplified Integer Problems and Results

The six simplified integer problems and the expressions for their probabilities have been listed as follows:

1. $P(C \mathbin{\widehat{+}} B_1 < 2^{m+1} \mid m, k) = 1 - 3(2^{-k-1}) + 2^{-m-1}$
2. $P(B_2 < A_2 \mid h, k) = 2^{-1} + 2^{-k-1} - 2^{-h}$
3. $P(C \mathbin{\widehat{+}} B_1 \mathbin{\widehat{-}} (A \gg h) < 2^{m+1} \mid m, h, k)$
   $= 1 - 3(2^{-k-1}) + 3(2^{-h-1}) - 2^{-m-h} + 2^{-m}$
4. $P(A_2 > 0 \mid h) = 1 - 2^{-h}$
5. $P(b_k \neq c_0) = 2^{-1}$
6. $P(a_h = 0) = 2^{-1}$.

The solution for the integer problem statement that we had derived from the floating-point inequality is included in our report. See our report for the full solutions to the simplified integer problems.

Methods

In order to solve the simplified integer problems, we produced a specific type of diagram that we refer to as a graph. These graphs can be used to determine the probability that a specific simplified integer problem is true. The following graph can be used to calculate the probability that the first simplified integer problem is true for the base case where $m = 3$ and $k = 1$:

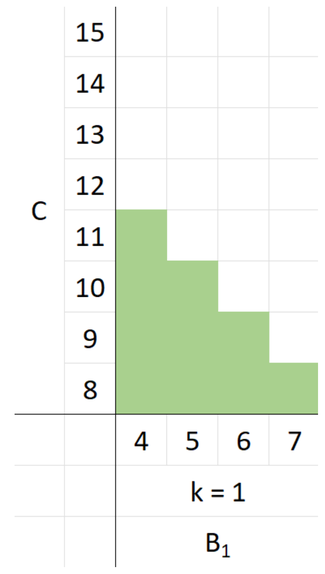

Figure 5: This is a graph that represents the base case for the first simplified integer problem where $m = 3$ and $k = 1$. Each box on the grid represents its corresponding C value added to its corresponding B_1 value. The green boxes represent the events where $C \mathbin{\widehat{+}} B_1 < 2^{m+1}$, while the white boxes represent the events where $C \mathbin{\widehat{+}} B_1 \geq 2^{m+1}$.

Using this graph, we can determine the probability that $C \mathbin{\widehat{+}} B_1 < 2^{m+1}$ where $m = 3$ and $k = 1$ as the number of green boxes divided by the total number of boxes present on the graph as follows:

$$P(C \mathbin{\widehat{+}} B_1 < 2^{m+1}) = P(C \mathbin{\widehat{+}} B_1 < 16) = \frac{10}{32} = \frac{5}{16}.$$

This is the fundamental method that we have used to solve the simplified integer problems in our project.

Conclusion

We have been working on this project for a duration of over 5 months and have certainly come a long way. Not only has this project expanded our mathematical skills, but it has also improved our collaboration skills. In conclusion, this project has made us passionate for a new field of computer science. We hope that we will have future opportunities to work on a project like this.

References

(1) OnlineMathLearning. *Binary Number System*. https://www.onlinemathlearning.com/binary-number-system.html [Accessed 24th February 2024].

(2) Maus P. *Bit Shift Calculator*. https://www.omnicalculator.com/math/bit-shift [Accessed 18th March 2024].