Department of Computer Science

Report

COMP36212-CW1

Tianpeng Huang

February 24, 2023

1 CPU and GCC

Some information about my PC:

- Model Name: MacBook Air
- CPU: Dual-Core Intel Core i5 a so-called bullet.
- GCC version: 4.2.1
- Processor Speed: 1.6 GHz
- Number of Processors: 1
- Total Number of Cores: 2
- L2 Cache (per Core): 256 KB

2 Part1-Implementation and testing of binary32 SR

2.1 Introduction

This part mainly requires us to implement float SR-alternative(double x) with code according to the given formula(Figure 1). Then do some experiments with the given SR function and SR-alternative. Verify that my SR-alternative(double x)implementation was successful. Observe which function accuracy performs better, and which one is more in line with the theoretical SR probability.

2.2 Implement the SR-alternative function

In this part I need to implement the following formula by code: The function SR alterna-

$$\mathrm{SR}(x) = \begin{cases} \mathrm{RA}(x), & \text{if } P < \frac{x-\mathrm{RZ}(x)}{\mathrm{RA}(x)-\mathrm{RZ}(x)} =: p, \\ \mathrm{RZ}(x), & \text{if } P \geq p, \end{cases}$$

Figure 1: SR Equation

tive takes a double-precision floating-point number x as input and returns a stochastically rounded 32-bit floating-point number. My idea of realizing this formula is this. At the beginning, I observed that the elements of this formula: x, RA(x), R(Z), P, and p are the key to solving the formula. I want to finish P and p first. In this way, the sizes of the two can be compared, and the different situations of rounding up and rounding down in SR can be realized.

The rand() function from the C standard library is used in the first step of the process to create a random number P between 0 and 1.

The technique then converts the two neighbouring 32-bit binary values to x using the C standard library's nextafterf() function. The two adjacent values are designated RZ (rounded-to-zero value) and RA (the rounded-away value). Next, use x, RA, RZ to calculate p. If P is less than p, the algorithm returns RA. Otherwise, it returns RZ.

Among them, regarding the positive and negative of x, the size of x in binary64 and the size of binary32, I accepted the inspiration from the textbook(Figure 2).

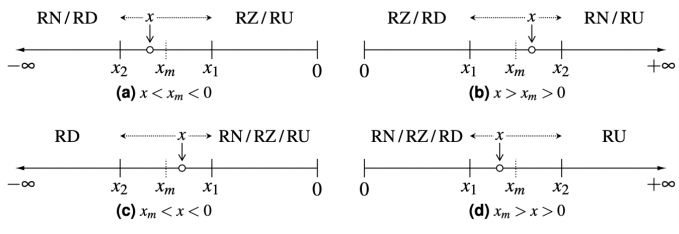


Figure 2: Rounding

2.3 Result Analysis

Figure 3 shows some experimental results in this part:

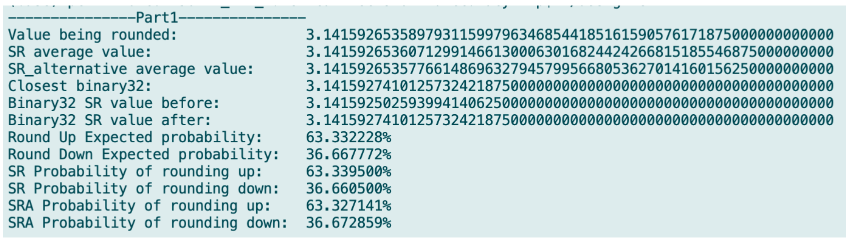

```
---------------Part1---------------
Value being rounded:              3.14159265358979311599796346854418516159057617187500000000000000
SR average value:                 3.14159265360712991466130006301682442426681518554687500000000000
SR_alternative average value:     3.14159265357766148696327945799566805362701416015625000000000000
Closest binary32:                 3.14159274101257324218750000000000000000000000000000000000000000
Binary32 SR value before:         3.14159250259399414062500000000000000000000000000000000000000000
Binary32 SR value after:          3.14159274101257324218750000000000000000000000000000000000000000
Round Up Expected probability:    63.332228%
Round Down Expected probability:  36.667772%
SR Probability of rounding up:    63.339500%
SR Probability of rounding down:  36.660500%
SRA Probability of rounding up:   63.327141%
SRA Probability of rounding down: 36.672859%
```

Figure 3: Part1 result

(Value being rounded) is the binary64 precision value of pi. Closest is the binary32 precision value of pi. We can use the nextafterf() function to calculate the neighboring binary32 values.

Next we calculate the average value of all the rounding operations for both SR functions. I'm adding up the rounded values for each of the 5000000 iterations and dividing by the total number of iterations. This way we can get the average. SR average value is 3.14159265360712991466130006301682442426681518554687 5, and SR-alternative average value is 3.141592653577661486963279457995668053627014160156 25. Comparing the two values with the binary64 value, we can find that the accuracy of the SR-alternative function is relatively high. There is not much difference between the two values, which proves the correctness of our SR-alternative function implementation.

In this experiment, we recorded the rounding up and rounding down of two SR functions so that the probability of their rounding can be calculated. The comparison with the theoretical probability can determine which SR function is more in line with the theoretical value.

The theoretical value of the rounding probability is 63.332228 up and 36.667772 down. The experimental result of SR rounding probability is 63.339500, down is 36.667772, 36.660500. The experimental result of SR-alternative rounding probability is 63.327141 and down is 36.672859. we can see that the rounding of SR is more in line with the theoretical value.

2.4 Absolute error of two SR function

The graph below shows the average of all rounding operations for the two SR functions. The absolute error of the mean of all rounded binary32 values for the two plotted rounding functions compared to the original binary64 as the rounding number i increases (sampled at a fixed step size, say every 1000 steps).

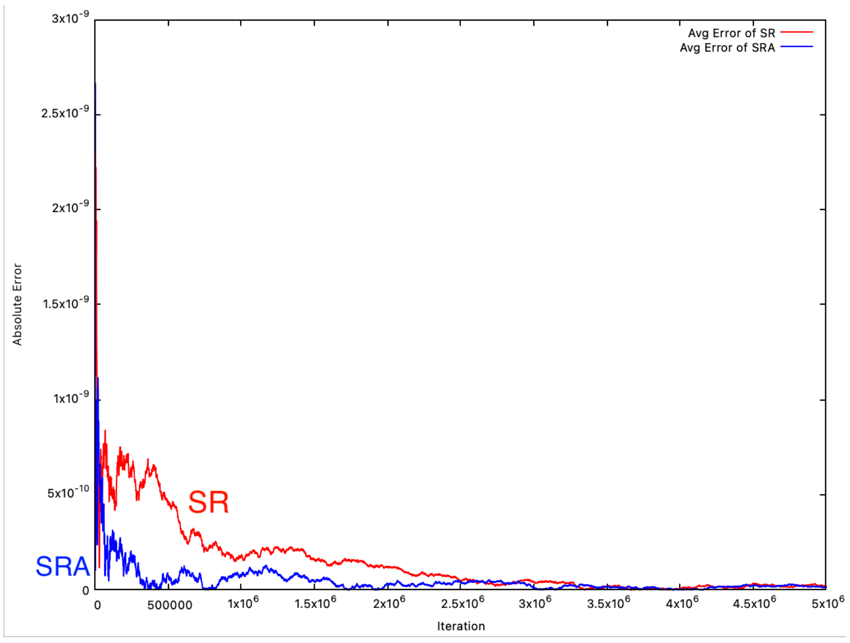


Figure 4: Plot average value of all the rounding

It can be clearly seen from the figure above that the red line represented by SR is obviously above the blue line represented by SR-alternative. This shows that the error of SR-alternative is smaller, and the accuracy of SR-alternative is higher than SR. This is also in line with the experimental conclusion we got in 2.3. Once again, it proves the correctness of my SR-alternative function implementation.

The basic SR function adds a random offset to the input value, which can cause rounding errors. In contrast, the "SR-alternative" method uses the "nextafter()" function to get the closest floating-point values, avoiding any rounding bias.

The basic SR function employs a fixed random offset, limiting the rounding precision. The "SR.alternative" function, on the other hand, computes the likelihood of rounding up depending on the difference between the input value and the two closest representable floating-point values. This results in more precise rounding and rounding.

When the number of iterations is very big, the difference between "SR" and "SR alternative" is almost minimal since the impact of rounding errors on the final result is negligible. The rounding errors associated with each individual computation compound and can become substantial as the number of repetitions grows. However, as the number of iterations rises, the difference in rounding error distributions between the standard SR and "SR alternative" approaches becomes less significant.

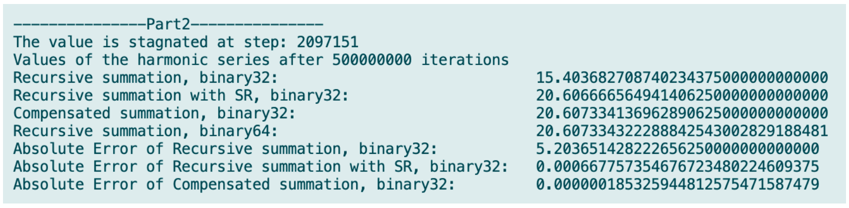
3 Part2-Stagnation and the Harmonic series

3.1 Introduction

This part introduction harmonic series will encounter the phenomenon of stagnation when computing, which leads to larger errors. we will use 3 methods to solve this problem. Conduct experiments and analysis.

3.2 Result Analysis

Figure 5 shows some experimental results in this part2:



```
---------------Part2---------------
The value is stagnated at step: 2097151
Values of the harmonic series after 500000000 iterations
Recursive summation, binary32:                          15.403682708740234375000000000000
Recursive summation with SR, binary32:                  20.606665649414062500000000000000
Compensated summation, binary32:                        20.607334136962890625000000000000
Recursive summation, binary64:                          20.607334322288842543002829188481
Absolute Error of Recursive summation, binary32:        5.203651428222656250000000000000
Absolute Error of Recursive summation with SR, binary32: 0.000667757354676723480224609375
Absolute Error of Compensated summation, binary32:      0.000000185325944812575471587479
```

Figure 5: Part2 result

In figure5, we can see that in binary32 recursive summation, after step 2097151, it has not been updated either. The final result is only 15.403682708740234375, which is very different from the true value. Stagnation phenomenon occurred.

The results show that the values of the three methods SR, Compensated summation, and binary64 are all more than 20. Among them, binary64(real value) performed best, followed by Compensated summation, and finally SR.

Finally, the results of the three solutions and the results of binary64 are used to calculate the absolute error. Compensated Summation was found to be the most accurate.

3.3 absolute errors of three harmonic series

Figure 6 plot the absolute errors of each of the three harmonic series at regular points (every 1000000 th iteration) from i = 1 to i = 500000000(five hundred million).

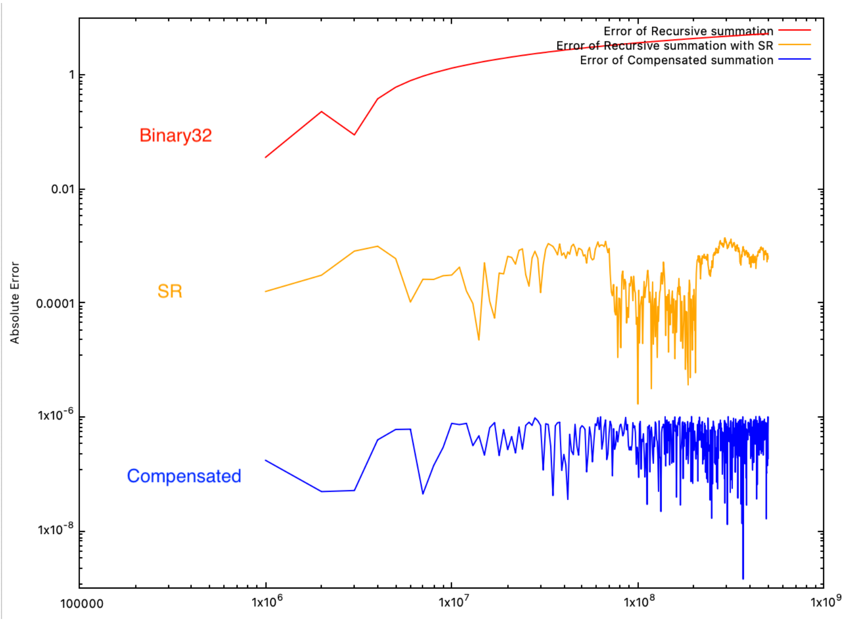


Figure 6: absolute errors of each of the three harmonic series

The top red line is the absolute error of binary64 summation and binary32 summation. It can be seen that the error is large, but the line is smooth. This is because binary32 uses a fixed number of bits to represent floating point numbers, which means it can only represent a limited number of values. When the partial sums of a harmonic series are very close to integers, the next term in the series may not be represented exactly using binary32, leading to rounding errors and loss of precision. In particular, it leads to stagnation, which is why later errors get bigger and bigger. Because binary32 is the default rounding mode is RN (), so the line is relatively smooth.

The orange line in the middle and the blue line at the bottom represent the absolute error between SR and binary64 and Compensated summation and binary64, respectively. The orange line is above the blue line, indicating that Compensated summation has a smaller error and greater accuracy than SR. One of the main advantages of compensated summation over stochastic rounding is that it can achieve higher accuracy with fewer computational resources. Compensated summation is a simple and efficient technique that works by keeping track of the error introduced during the computation and compensating for it at each step. In contrast, stochastic rounding requires generating random numbers, which can be computationally expensive, and introduces additional noise into the computation.

One of the main advantages of compensated summation over stochastic rounding is that it can achieve higher accuracy with fewer computational resources. Compensated summation is a simple and efficient technique that works by keeping track of the error introduced during the computation and compensating for it at each step. In contrast, stochastic rounding requires generating random numbers, which can be computationally expensive, and introduces additional noise into the computation. This explains why the SR lines fluctuate more.

4 Part3-Stagnation and Matrix Operations

4.1 Introduction

Matrix Operations is a fundamental operation in many areas of computer science and engineering, such as machine learning, graphics and image processing, scientific computing, database management, etc.

Rounding errors can occur when working with floating-point numbers when computation, which are approximations of real numbers. Like part2, there will also be stagnation when there is a large difference between the two numbers. These errors can accumulate during Matrix Operations, leading to significant inaccuracies in the final result.

Stochastic rounding and compensated summation can help to reduce the impact of these errors. By using SR during matrix multiplication, we can introduce some randomness into the computation and help to reduce the impact of rounding errors. This can lead to more accurate results and better performance, especially when working with large matrices and complex algorithms.

4.2 Implement

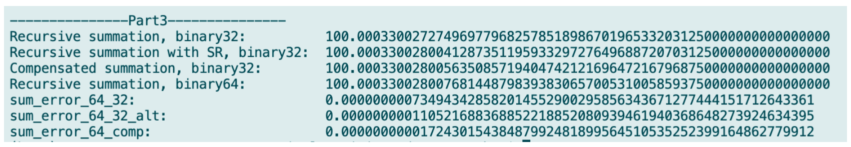
First I will initialize matrices of two squares.(A[N][N] and B[N][N]) whose values will stagnate during the operation. Then use matrix-multiply() to perform matrix multiplication. Because there will be stagnation, then we use the four methods mentioned in Part2 to calculate the value of each element in the matrix, conduct experiments, and analyze errors(C-32[N][N], C-32-alt[N][N], C-comp[N][N], C-64[N][N]).

Next, I want to enlarge error, so I will calculate the total value of the matrix in 4 matrices just like Part2. This numerical difference will be obvious. Due to stagnation, precision, random rounding, the 4 results will not be the same.

Absolute error for each of the three matrices from N*N = 1*1 to N*N = 100*100 versus the binary64 matrix. Save it to a txt file for easy drawing of images later. Finally, calculate the sum of the absolute errors of the three matrices and the binary64 matrix. Explore which method is better.

4.3 Result Analysis

Figure 7 shows some experimental results in this part3:



```
---------------Part3---------------
Recursive summation, binary32:          100.00033002727496977968257851898670196533203125000000000000000000
Recursive summation with SR, binary32:  100.00033002800412873511959332972764968872070312500000000000000000
Compensated summation, binary32:        100.00033002800563508571940474212169647216796875000000000000000000
Recursive summation, binary64:          100.00033002800768144879839383065700531005859375000000000000000000
sum_error_64_32:                        0.000000000734943428582014552900295856343671277444151712643361
sum_error_64_32_alt:                    0.000000000110521688368852218852080939461940368648273924634395
sum_error_64_comp:                      0.000000000017243015438487992481899564510535252399164862779912
```

Figure 7: Part3 result

In Figure 5, we can see that the 4 results have numerical differences. Among them, as part2 said, binary64 is close to the true value. Binary64 is numerically the largest, followed by Compensated summation, followed by SR, and finally binary32. This means that Compensated summation is closer to the true value, followed by SR.

Finally, the absolute error is calculated using the results of the three solutions and the results of binary64. Compensated summation was found to be the most accurate, followed by SR.

4.4 Absolute errors of three matrix

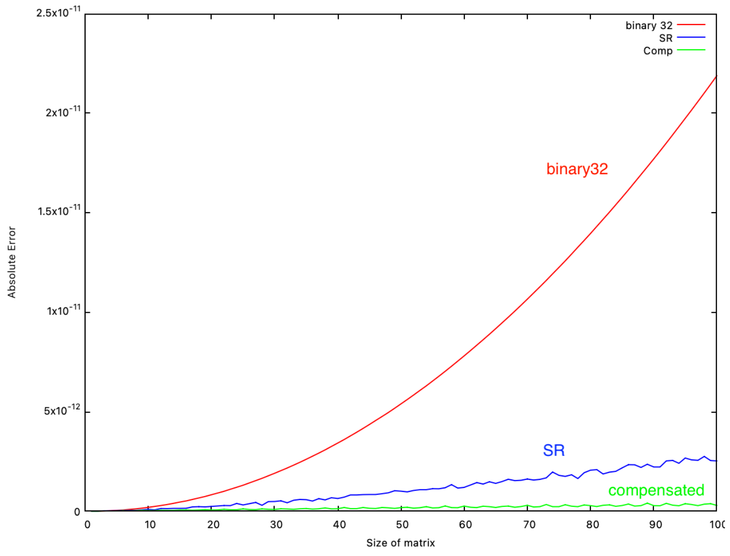


Figure 8: Absolute errors of three matrix

The top red line is the absolute error of binary64 summation and binary32 summation. It can be seen that the error is large, but the line is smooth. The blue line in the middle and the green line at the bottom represent the absolute error between SR and binary64 and Compensated summation and binary64, respectively. The blue line is above the green line, indicating that Compensated summation has a smaller error and greater accuracy than SR. When N¡10, the difference is not obvious. Because the matrix was small at that time, the error caused by the operation was also small. The specific reason I have explained in **3.3 of Part2**, I will not repeat the explanation here.

4.5 Summary

In matrix operations, it is true that Compensated summation and SR help us obtain accurate operations. Finally, compensated summation is a widely used technique in numerical analysis and has been studied extensively. There are many established algorithms and software libraries that implement compensated summation and have been shown to be effective in a wide range of contexts. In contrast, is a relatively new technique that is still being studied, and there is less established software and algorithms available for its use.

Overall, while both compensated summation and stochastic rounding can be effective in improving the accuracy of computations involving the Harmonic series, compensated summation is generally preferred due to its higher accuracy, lower computational cost, deterministic nature, and more established.

References:

1. M. P. Connolly, N. J. Higham, and T. Mary, Stochastic rounding and its probabilistic backward error analysis, SIAM J. Sci. Comput., 43 (2021), pp. A566–A585.

2. Zhang, S.Q., McDanel, B. and Kung, H.T., 2022, April. Fast: DNN training under variable precision block floating point with stochastic rounding. In 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA) (pp. 846-860). IEEE.

3. Croci, M., Fasi, M., Higham, N.J., Mary, T. and Mikaitis, M., 2022. Stochastic rounding: implementation, error analysis and applications. Royal Society Open Science, 9(3), p.211631.

4. Graillat, S., Jézéquel, F. and Picot, R., 2015. Numerical validation of compensated summation algorithms with stochastic arithmetic. Electronic Notes in Theoretical Computer Science, 317, pp.55-69.

5. Evstigneev, N.M., Ryabkov, O.I., Bocharov, A.N., Petrovskiy, V.P. and Teplyakov, I.O., 2022. Compensated summation and dot product algorithms for floating-point vectors on parallel architectures: Error bounds, implementation and application in the Krylov subspace methods. Journal of Computational and Applied Mathematics, 414, p.114434.

6. Cheng, J., Zhou, H. and Dong, J., 2021. Photonic matrix computing: from fundamentals to applications. Nanomaterials, 11(7), p.1683.

7. Jou, J.Y. and Abraham, J.A., 1986. Fault-tolerant matrix arithmetic and signal processing on highly concurrent computing structures. Proceedings of the IEEE, 74(5), pp.732-741.