COMP36212 EX1
Testing and Applications of Stochastic Rounding

2023

Introduction

Most processors which include floating-point (FP) arithmetic provide support for all the rounding modes required by the IEEE 754 standard [4]. The default rounding mode is most commonly round-to-nearest ties-to-even (RN), which rounds all the values to the nearest FP number. A non-standard rounding routine introduced in Week 3 of this module, *stochastic rounding* (SR), is therefore not straightforward to test on these systems, and either requires making a prototype hardware design with SR, or simulating SR in software. This assignment explores the latter, simulating SR (of binary32 arithmetic) in software and analysing its behaviour when applied to basic applications.

Stochastic rounding

Stochastic rounding was introduced in Week 3, where it was shown to be a non-deterministic rounding mode which can help to cancel out rounding errors over many rounding operations. An alternative definition of stochastic rounding is

$$\mathrm{SR}(x) = \begin{cases} \mathrm{RA}(x), & \text{if } P < \frac{x-\mathrm{RZ}(x)}{\mathrm{RA}(x)-\mathrm{RZ}(x)} =: p, \\ \mathrm{RZ}(x), & \text{if } P \geq p, \end{cases} \tag{1}$$

where x is some real value, RA stands for round-away-from-zero, RZ is a rounding mode round-toward-zero which was introduced in Week 2, and $P \in [0, 1)$ is a random number from a uniform distribution. Note that in Equation 1, $x - \mathrm{RZ}(x)$ is the distance between x and a floating-point number on the left of it (on the negative side of the axis it would be right of it), and $|\mathrm{RA}(x) - \mathrm{RZ}(x)|$ is the distance between the two floating-point values that surround x.

Part I: Implementation and testing of binary32 stochastic rounding (8 marks)

The C program provided for this assignment contains an implementation of the stochastic rounding operation based on binary manipulation, which can be used to verify your results. This implementation is provided by the function `float SR(double x)`, and takes as inputs binary64 numbers and produces binary32 numbers which are rounded stochastically. Special cases such as overflow are not covered here for simplicity.

The code can be compiled by running `gcc -o assignment1 assignment1.c -lm` in a linux terminal, and run with `./assignment1`. The flag `-lm` may be required on some compilers to use `math.h` functions. An

empty function body is also provided provided, `SR_alternative`, which should be completed during Part I.

For this part of the assignment you are required to:

1. Implement your own version of stochastic rounding according to Eqn. (1), defining the operations within the function `float SR_alternative(double x)`, rounding from binary64 to binary32.

2. Discuss in the report the key steps involved, and how these are implemented in software to achieve stochastic rounding.

3. Test your function to demonstrate that it correctly implements Eqn. (1). To do so, pick a value that is representable in binary64 more accurately than in binary32, for example π (e.g. `M_PI`, which is predefined in C), or 1/3, and round it 5 million times. You should compare results using both your implemented function and the provided `SR` implementation in order to validate your implementation.

4. Count the number of times your implemented function rounded up and down, and use these counters to confirm that the observed probabilities of rounding up and down are close to expected. To calculate the expected probabilities round the binary64 value up and down (e.g. using `nextafterf()` function available in C) to obtain the two neighbouring binary32 values, then compute the distances of each of them to the binary64 value, and divide by the distance to each other. You may also find the standard C function `fabs()` useful here.

5. Calculate the average value of all the rounding operations for both the provided and your implemented `SR` functions. Plot the absolute error of the average of all the rounded binary32 values for both the rounding functions compared with the original binary64 as the number of roundings i increases (sample at fixed steps such as on every 1000th step). Have rounding iteration i on the horizontal axis and errors on the vertical axis. Use your preferred method to save the data (e.g. print out to terminal, or write on a file), but making this clear in the code submission. Similarly, plot the data using any method you prefer (e.g. `gnuplot`, `pgfplots`), but ensure to include the plots in the final report. Analyse and discuss in the report text how the values change as more roundings are performed, including how they change in relation to the unrounded binary64 value, whether the values obtained from `SR` and `SR_alternative` are consistent, and whether findings are as expected.

Reports will be marked according to the rubric in Table. 1, with the submitted code used to help verify your implementation and testing approach (e.g. to ascertain correctness of the `SR_alternative` function implementation, and the code used to compute expected and observed rounding probabilities and gather data).

While code style is not being assessed, it's suggested to use a concise style, with *descriptive variable names* and *informative comments*, lines of code *no longer than 79 columns*, and no commented out (testing) code should appear in the submitted version unless it is useful code for printing out some data for example.

Part II: Stagnation and the Harmonic series (8 marks)

Stagnation occurs in floating-point arithmetics when for some FP values a and b, where $a >> b$, $\mathrm{RN}(a + b) = a$. It can occur in any application that uses FP numbers, but is most easily shown with a toy problem called *harmonic series*, introduced in Week 3. It is defined as

$$\sum_{i=1}^{\infty} 1/i = 1 + 1/2 + 1/3 + ...$$

and in infinite precision is known to be a *divergent series*, but converges to some value in limited precision computer arithmetics. To analyse this on computers, we use a truncated version of the series:

$$\sum_{i=1}^{N} 1/i = 1 + 1/2 + 1/3 + ... + 1/N. \tag{2}$$

In this exercise we will be analysing what happens to this series in different arithmetics, rounding modes, or summation methods. You may find [5] to be a useful research paper to have a look at for this part.

In the same C file used for Part I, implement the harmonic series and analyse it. A recursive summation method with binary32 arithmetic and round-to-nearest mode (default) in the addition and division operations, is provided as a starting point.

1. Use recursive summation with binary32 arithmetic, but simulate stochastic rounding. To do this, compute division and addition in binary64 arithmetic, and round the result using your implemented `SR_alternative` function (from Part I), on each iteration, to update the binary32 result `fharmonic_sr`.

2. Use compensated summation with binary32 arithmetic and round-to-nearest mode. (Hint: this will require the `fastTwoSum` function. You may want to look at code from the Week 3 examples).

3. Use recursive summation with binary64 arithmetic and round-to-nearest (default). While all the three previous approaches store the result in binary32, here store it in binary64. This acts as a reference for comparison since it is expected to be the most accurate result.

4. Run the harmonic series for $N = 500000000$ (five hundred million) iterations. Report at what i step the basic binary32 implementation with recursive summation stagnates. Additionally, compute the absolute errors of the three binary32 harmonic series values compared with the binary64 one, and report which one of the three has the lowest error and which one the highest.

5. Plot the absolute errors (take absolute values of absolute errors to make all errors positive) of each of the three harmonic series at regular points (every 1000000th iteration) from $i = 1$ to $i = 500000000$ (five hundred million). The horizontal axis should contain i, while the vertical axis the errors of the harmonic series. Set both axes to *logarithmic scale*. Once again, feel free to use any method you prefer to save and plot the data, but ensure to include this in your report.

As in Part I, reports will be assessed according to the rubric in Table. 1, with the submitted code used to help verify your implementation and testing approach.

Part III: Other Applications of SR (8 marks)

Demonstrate another algorithm or application where SR can improve performance, and evaluate the effectiveness of SR, e.g. by analysing performance with the four arithmetics used in Part II. Consider research papers, such as [5, 3, 1, 2], and other research sources for inspiration, and document in your report the method, results and analysis following the format of Part II. The awarded marks for Part III are the same as Parts I & II, therefore you should aim to invest similar effort for this section.

Part IV: The report (6 marks)

Write a (research) report that contains your analysis of the implementation of stochastic rounding, and the findings of applying stochastic rounding in computing the infinite series (Part II) and your alternative

application (Part III). Include some basic info about your CPU, and the version of your `GCC` compiler (or other) used for compiling the C code. Note that numerical results can have slight differences depending on whether the 80-bit arithmetic is enabled on Intel CPUs, or whether `FMA` instructions are enabled.

Submit the report as a single zip file `your_name.zip`, containing the report as a single PDF file, and your modified `assignment1.c`. It is recommended to write your report in LaTeX, however this is not a strict requirement and other software can be used to produce the report PDF.

Marks for the report will be given for the *clarity* and *conciseness*, as well as the correctness and the level of detail of the content (see Tab. 1). The report should be spellchecked and well-presented, with diagrams fully readable when printed out. Taking into account black and white printing requires using different shapes and dashed/dotted patterns to compensate for the loss of colour. Finally, the report should be structured into sections and subsections matching the assignment parts, and **no longer than 6 pages**, excluding references.

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] M. Croci, M. Fasi, N. J. Higham, T. Mary, and M. Mikaitis, *Stochastic rounding: Implementation, error analysis, and applications*, MIMS EPrint 2021.17, Oct. 2021. Revised Jan. 2022.

[3] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, *Deep learning with limited numerical precision*, in Proceedings of the 32nd International Conference on Machine Learning, F. Bach and D. Blei, eds., vol. 37 of Proceedings of Machine Learning Research, Lille, France, July 2015, PMLR, pp. 1737–1746.

[4] *IEEE Standard for Floating-Point Arithmetic, IEEE Std 754-2019 (revision of IEEE Std 754-2008)*, Institute of Electrical and Electronics Engineers, Piscataway, NJ, USA, July 2019.

[5] D. Malone, *To what does the harmonic series converge?*, Irish Math. Soc. Bull., (2013), pp. 59–66.

Table 1: Marking rubric for COMP36212-EX1 (to achieve a given level, all attributes/skills to the left must also be demonstrated).

Section	Task	0-25%	26-50%	51-75%	76-100%
Part I (8 marks)	Implement stochastic rounding (SR) in binary32, demonstrating appreciation of the methods used and underlying theory.	Demonstrate understanding of SR and its theoretical underpinning, and correct use of the provided SR function.	Implement correctly your own version of SR, demonstrating understanding of how to gather data and plot results/analyse performance.	Test SR in practice, providing in-depth discussion, analysis and validation of results/performance	Exceptional implementation, analysis, discussion and validation, going beyond the scope of the assignment brief, and drawing on the scientific literature.
Part II (8 marks)	Implementing and testing arithmetics in computing the harmonic series.	Demonstrate understanding around the issue of stagnation in computer arithmetic, and how SR can be used to overcome it	Implement and test a range of summation methods and a reference computation, and report their performance	Demonstrate correctness of the implementation/experiments, providing in-depth discussion, analysis and validation of results/performance.	Exceptional implementation, analysis, discussion and validation, going beyond the scope of the assignment brief, and drawing on the scientific literature.
Part III (8 marks)	Implementing and testing arithmetics in other applications.	Analogous to Part II.	Analogous to Part II.	Analogous to Part II.	Analogous to Part II.
Part IV (8 marks)	Writing of a research report summarising the methods and implementations used for the above tasks, and analysing the results.	Some correct results from Parts I/II/III are present in the report, with appropriate presentation and length.	Report follows well-defined structure, mapping clearly to the assignment brief. Figures, equations and tables are produced to a high standard and referenced accordingly	The report provides substantial analysis/discussion of the experimental data, with appropriate references to scientific literature.	The report provides exceptional analysis beyond the scope of the requirements and is based on and cites scientific literature.