Stochastic Rounding: Algorithms and Hardware Accelerator

Mantas Mikaitis
Department of Mathematics
The University of Manchester
Email: mantas.mikaitis@manchester.ac.uk

Abstract—General algorithms and a hardware accelerator for performing stochastic rounding (SR) are presented. The main goal is to augment the ARM M4F based multi-core processor SpiNNaker 2 with a more flexible rounding functionality than is available in the ARM processor itself. The motivation of adding such an accelerator in hardware is based on our previous results showing improvements in numerical accuracy of ODE solvers in fixed-point arithmetic with SR, compared to standard round-to-nearest or bit truncation rounding modes. Furthermore, performing SR purely in software can be expensive, due to requirement of a pseudorandom number generator (PRNG), multiple masking and shifting instructions and an addition operation. Also, saturation of the rounded values is included, since rounding is usually followed by saturation, which is especially important in fixed-point arithmetic due to a narrow dynamic range of representable values. The main intended use of the accelerator is to round fixed-point multiplier outputs, which are returned unrounded by the ARM processor in a wider fixed-point format than the arguments.

Index Terms—stochastic rounding, fixed-point arithmetic, floating-point arithmetic, bfloat16

I. INTRODUCTION

SpiNNaker is a 18-ARM968-core (integer only) chip for simulating neural networks, including ordinary differential equations (ODEs) of neurons [1]. Our previous work [2] explored numerical accuracy issues in ODE solvers run in fixed-point arithmetic with the main conclusion that rounding errors are a major factor in divergence of the solution from the reference double precision solution and that SR can help in reducing divergence. We also showed that fixed-point with SR is \(2.6 - 4.2\times\) faster than emulating floating-point arithmetic in software.

The next generation SpiNNaker, SpiNNaker 2 will be based on an ARM Cortex-M4F processor [3], which does not have a capability of rounding a fixed-point number to a specified number of bits. There are three instructions with rounding available: SMMLAR - multiply two numbers, add a third number to the top 32-bits of the result and return the rounded 32 top bits; SMMLSR - the same as SMMLAR, but subtract the third argument; SMMULR - multiply and return the rounded 32 top bits of the result [4]. Rounding is done by adding \(0x80000000\) to the product, therefore the tie-breaking rule is round-up [5]. While this would work well for \(s16.15 \times u0.32\) multiplications (where \(\{s/u\}\) defines the number of integer bits and \(Y\) fractional bits), it is limited in terms of other different mixed-format multiplications demonstrated in [2]. To implement RN and stochastic rounding would require multiple instructions, usually working on two registers containing a 64-bit unrounded value. Furthermore, there is no mention as to whether there are instructions available on this processor to perform saturation after rounding (return a maximum representable value on overflow). While saturation instructions for 32-bit values with configurable saturation bit position and saturated addition are available on the M4F, saturating a 64-bit value from the multiplication would need to be done by comparison and because it is a value across two registers, multiple instructions would be required to obtain a rounded and saturated value somewhere in the middle of a 64-bit value. Additionally, since ARM M4F has a single-precision floating-point (fp32) unit, it is beneficial to add fp32 to bfloat16 (fp32 with the bottom 16 bits removed) rounding, which is an elegant format for storage and can be operated on using fp32 hardware.

The contributions of this paper are:

- Two bit level algorithms for doing stochastic rounding and saturation (Section II).
- The architecture of the accelerator for doing rounding and saturation (Sections IV and V).
- Three accelerators with 8/16/32-bit random number precisions in stochastic rounding are evaluated in 22nm technology. Leakage and area comparisons are demonstrated (Section VI).

II. ALGORITHMS

Stochastic rounding has recently been explored in machine learning due to substantial improvements in reducing rounding errors in low precision numerical formats [6], [7], [8], [9]. The first mention of it can be traced back to [10].

Stochastic rounding differs from the standard rounding mode round-to-nearest in that instead of always rounding to the nearest number, the decision about which way to round is non-deterministic and the probability of rounding up is proportional to the residual (trailing bits that do not fit into the destination format). Given a real number \(x\), an output fixed-point format to round the value to, \(<s, i, p>\) (where \(s\) tells us whether it is signed or unsigned format, \(i\) defines the number of integer bits and \(p\) defines the number of fractional bits); defining \(\lfloor x \rfloor\) as the truncation operation (cancelling a number of bottom bits with weights smaller than \(2^{-p}\) and leaving...
A standard check for overflow at both ends of the dynamic algorithms require 5 operations in the main rounding parts right). Stochastic rounding by addition looks shorter, but both (in the algorithms ≪ and ≫ stand for binary shifts left and right). Stochastic rounding by addition looks shorter, but both algorithms require 5 operations in the main rounding parts (saturation is the same in both cases). Saturation logic is a standard check for overflow at both ends of the dynamic range and note that if the input and output numbers would be unsigned, we would have to perform only one comparison instead of two. Also it is worth noting that round-to-nearest can be implemented similarly to Algorithm 2 as shown in Algorithm 3.

By comparing Algorithms 2 and 3 notice that SR has an overhead of a PRNG plus one operation to mask off the top bits of the random number, compared with round-to-nearest. Depending on the optimization level and whether rounding is inlined or not, these algorithms might be compiled for a fixed n and therefore optimized on compilation for specific use cases.

### Algorithm 2: Stochastic rounding by addition

```plaintext
function SATSR_INT64_INT32(X, n)
    P ← PRNG32()
    P ← P & ((1 ≪ n) - 1)
    X ← X ≫ n
    if X > MAX_INT32 then
        return MAX_INT32
    if X < MIN_INT32 then
        return MIN_INT32
    return X
```

### Algorithm 3: Rounding to nearest with round up on a tie

```plaintext
function SATRN_INT64_INT32(X, n)
    X ← (X + (1 ≪ (n - 1))) ≫ n
    if X > MAX_INT32 then
        return MAX_INT32
    if X < MIN_INT32 then
        return MIN_INT32
    return X
```

### III. Numerical experiments

The algorithm of choice for the proposed hardware accelerator is Algorithm 2 and here we test it first in software simulation on an ARM968 processor using fixed-point arithmetic. The main advantages of SR can be detected in summation algorithms, with the data with rounding errors biased into one direction which dominates the final error in the result of the sum. Following the approach taken by [11] we have applied the Algorithm 2 of stochastic rounding in software on a basic recursive summation algorithm evaluating the harmonic series; this series is a divergent series but converges when implemented in limited precision arithmetic as a simple recursive summation. The series is defined as $\sum_{i=1}^{\infty} \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots -$ it can be seen that the addends are getting smaller while the total sum keeps increasing and as [11] reported the sum converges in floating-point arithmetic when the addends become small enough that they do not change the total sum anymore (due to very different exponents and round off error on addition). This issue is called stagnation in [12], a problem which happens in summing algorithms in floating-point arithmetic.

This experiment was run in 32- and 16-bit fixed-point arithmetics. The sum has a numerical type of s16.15 or s8.7
and is initialized to 1. Then the series is started from \( i = 2 \) and the division is done in either 32-bit or 16-bit fractional type \( u0.32 \) or \( u0.16 \) and the addend is rounded to the sum’s format with various rounding routines. While fixed-point addition is known to be exact, in this case it is not since the addends have more fractional precision than the sum.

Table I demonstrates the results with various fixed-point types; floating-point results are also provided for comparison. Five million iterations were chosen to have a manageable run time, but the number of iterations to convergence is also reported. As expected, most of the fixed-point types converge as soon as the addends in the series become small enough to be evaluated at lower than \( s16.15 \) precision, when the values cross 0.5e. However, it can be seen that fixed-point with SR can accurately replicate the sum of the fp64 type in 5 million iterations without converging. Given that stochastic rounding is probabilistic rounding, it might still produce some effect in later iterations stochastically and therefore it can be said that it never converges, i.e. there is a diminishing, but non zero probability of rounding up the addends and affecting the sum.

In practice it converges also when the numerical type of the addends runs out of bits and the probability of rounding up becomes 0. This can also happen if there is a limited amount of random bits available for performing stochastic rounding and can especially be significant in rounding the double precision adder/accumulator results as these can be held in thousands of bits before rounding [13]. This gives us a confirmation that Algorithm 2 works as expected.

In summary, running the harmonic series 50 times in \( s16.15 \) with SR, it is shown that the averaged result has a very small error compared to the sum done in FP64, while \( s16.15 \) with RN stagnates just after 65536 steps.

V. DESIGN

In this section we describe the specification of the proposed accelerator that we have designed and included in the upcoming SpiNNaker 2 neuromorphic chip [3].

The rounding and saturation accelerator is a memory mapped unit, connected through an AHB bus - a set of memory addresses are allocated for different rounding routines and numerical formats to which arguments are written and from which the rounded values are read out. For rounding multiplication results, it is useful to have a \( 64bit \rightarrow 32bit \) number rounding, with configurable rounding bit position from 0 to 31. Given that the ARM M4F processor has 32-bit wide interfaces, two memory cycles are required for inputting 64-bit arguments through AHB into the accelerator. For other use cases, \( 32bit \rightarrow 32bit, 32bit \rightarrow 16bit \) and \( 16bit \rightarrow 16bit \) round and saturate are also supported. For this, one memory cycle is required for input, therefore, aiming at single cycle for the main part of rounding, the accelerator will either have a 4 or 3 cycle delay for a write-round-read operation for 64- or 32-bit arguments respectively. Both signed and unsigned number types are supported, given a wide range of use cases for both types shown in [2]. Furthermore, as the ARM M4F has single precision floating-point hardware support, it might be beneficial to round fp32 to bfloat16 (16-bit single precision floating-point format with sign, 8 exponent and 7 significant bits). This format can be useful for representing and storing neural network weights for example, which can be operated on using the floating-point unit by inputting into a higher part of the floating-point register and then rounded back to bfloat16 before writing to memory. Finally, given that an adder is required in stochastic rounding, we can also add round-to-nearest (round up on ties, for cheaper implementation), which can reuse the adder to add 0x1 shifted to the required rounding bit position.

### Table I

| Arithmetic | Sum at \( i = 5 \times 10^6 \) | Error at \( i = 5 \times 10^6 \) | Iterations to converge |
|------------|-------------------------------|-------------------------------|------------------------|
| FP64       | 16.002                        | 0                             | \( 2.81... \times 10^{14} \) |
| FP32       | 15.404                        | 0.598                         | 2007152                |
| FP16       | 7.086                         | 8.916                         | 513                    |
| \( s16.15 \) RN | 11.938                        | 4.064                         | 65537                  |
| \( s16.15 \) RD | 10.553                        | 5.449                         | 32769                  |
| \( s8.7 \) RN | 6.414                         | 9.588                         | 257                    |
| \( s8.7 \) RD | 5.039063                      | 10.963                        | 129                    |
| \( s16.15 \) SR \[50 runs\] | Mean = 16.002                 | \(-0.000135765\)              | \( 2^{32} + 1 \)       |
| \( s8.7 \) SR \[50 runs\] | Mean = 11.205                 | 4.797                         | \( 2^{16} + 1 \)       |

In this section we describe the specification of the proposed accelerator that we have designed and included in the upcoming SpiNNaker 2 neuromorphic chip [3].
supplied by the AHB bus, depending on which address was written by the processor.

The main mechanism is to pick the top 32 bits of the residual depending on the configuration register, which is set up beforehand and contains the number of bits to round, 0 to 31 (0 means round 1 bit, 31 means round 32 bits). Then, the 32 bits after these residual bits are also isolated which is an unrounded result at this point. The minimum number of bits to round is 1, therefore the data input is extended to the right by 31 bits to support Verilog’s base minus 32 bit slicing functionality. For the same reason the input data is extended by 32 bits to the left, for overflow detection by operating on 32 bits after the result.

A pseudorandom number is added to the residual and the carry bit c_out is captured from that. Then, depending on the round mode, either the top bit (in case of round-to-nearest-up) or the c_out (in case of stochastic rounding) is added to the unrounded result which performs round-up if it is 1 and round-down if it is 0. Finally the rounded result and the overflow bits are used to saturate the result if required.

VI. Evaluation

The main logical path of the accelerator contains two adders - one 32-bit for rounding and one 8-, 16- or 32-bit for the stochastic rounding part when a random number is added to the residual. The architectural diagram in Figure 1 demonstrates a 32-bit version, but it is worth evaluating the three versions as there is some evidence that not all of the 32 bits are needed in SR, as shown in our previous work [2]. All of the logic, except some saturation checks, are performed in a single cycle.
Saturation logic contains basic checks of the overflow flags depending on the address input from the AHB bus and in our implementation is done on the AHB output cycle.

Following the synthesis study approach taken by us before [14], we have executed it on the current accelerator using the makeChip hosted design service platform [15] for the GLOBALFOUNDRIES 22FDX technology [16] for which the SpiNNaker 2 chip is being developed. An ultra-low voltage 8T-CN RX standard-cell library with multiple voltage threshold options is used for implementation. The standard cells use the adaptive body biasing (ABB) technique for post-silicon adaptation of transistor threshold voltage [17], [18]. Namely, two main categories of cells are used: Low-Voltage-Threshold (further called LVT) and Super-Low-Voltage-Threshold (further called SLVT) cells - the former with the larger propagation delay but significantly less leakage than the latter, much faster, cells. A nominal supply voltage of 0.50V is considered for low power operation. Due to manufacturing variations, synthesis is performed in a worst case speed condition at 0.45V and −40 °C. Three versions of the accelerator are synthesized varying the clock frequency constraint $f_{clk} = \{50, 100, 150, 200, 250, 300, 350, 400\}\text{MHz}$ and leakage and area is measured.

Figure 2 shows the area comparison of the three accelerators for different clock constraints and Figure 3 shows leakage. From this data it can be seen that at low clock frequencies, the adder width can save some area and leakage (more than an order of magnitude less leakage with 8-bit SR at $f = 150\text{MHz}$), but at higher frequencies other costs dominate and the savings are not that evident anymore. Especially for leakage; the leakage of the circuit apart from the adder dominates the total and changing to a smaller adder does not produce significant improvements.

Figure 4 shows the rounding accelerator highlighted in a layout of a single PE. The area of the accelerator is estimated at 1004 $\mu\text{m}^2$.

VII. Conclusion

We have presented algorithms and accelerator design for doing rounding and saturation of numbers up to 64 bits, including stochastic rounding which is becoming popular in machine learning. This includes rounding of fixed-point/integer values at any bit position as well as fp32 to bfloat16 rounding. The chosen SR algorithm was tested on the harmonic series done with a basic recursive summation, demonstrating how SR can help avoid numerical stagnation in fixed-point arithmetic. Evaluation of the accelerators with different precisions of SR step was performed, showing an order of magnitude of leakage improvement with 8-bit SR at $f = 150\text{MHz}$. The accelerator will be included in the SpiNNaker 2 chip, which is scheduled for 2020 release and is based on an ARM Cortex-M4F processor. Since this processor does not provide a wide array of rounding and saturation instructions to support fast fixed-point arithmetic, especially mixed-format fixed-point arithmetic, this accelerator will complement it and provide that functionality.

The presented results should also be applicable in implementing stochastic rounding of floating-point arithmetic, such as rounding the extended precision results from the floating-point adder or multiplier [19], [20].

VIII. Acknowledgements

Many thanks to Sebastian Höppner, Stefan Scholze and Andreas Dixius of Technical University of Dresden for the help with the synthesis tools. The work was funded by the Kilburn studentship and an EPSRC Doctoral Prize Fellowship.

References

[1] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, “The SpiNNaker Project,” Proceedings of the IEEE, vol. 102, no. 5, pp. 652–665, May 2014.
[2] M. Hopkins, M. Mikaitis, D. R. Lester, and S. Furber, “Stochastic rounding and reduced-precision fixed-point arithmetic for solving neural ODEs,” Accepted for the Philosophical Transactions of the Royal Society A.
[3] S. Höppner and C. Mayr, “SpiNNaker2-Towards Extremely Efficient Digital Neuromorphics and Multi-scale Brain Emulation,” in 2018 NICE Workshop, 2018. [Online]. Available: niceworkshop.org/2018-nice-workshop/
[4] ARM, “ARM Cortex-M4 Processor, Technical Reference Manual,” 2015.
[5] ——, “Cortex-M4 Devices, Generic User Guide,” 2010.
[6] M. Höfling and S. E. Fahlman, “Probabilistic rounding in neural network learning with limited precision,” Neurocomputing, vol. 4, no. 6, pp. 291 – 299, 1992. [Online]. Available: http://www.sciencedirect.com/science/article/pii/092523129290014G
