Two Alternative Approaches To Rounding Issues In Precision Computing With Accumulators, With Less Memory Consumption

Roy P. Gulla, Jr.

***Abstract*— Recent developments in numerical formatting have introduced a new system and a new emphasis on the use of accumulators for numerical computation. There also has been a recent development in this Posits numerical system, designed by John Gustafson, utilizing logarithmic bases. Here an alternative in light of these new developments is presented in a way to incorporate a design feature of Gustafson's format which negates the need for fractional bits in his system. One possible hardware design for implementation of the formatting is also briefly mentioned.**

I. INTRODUCTION

In recent computation the IEEE float type has fallen short with its approach to precision and subsequently various methods have been employed to maximize the exactness of representations of numbers, including those most recently introduced by John Gustafson's posit system. None of these methods are easy on memory consumption, and memory constraints on such number formats and the space allotted for them are not considered by most supercomputing systems as a high priority, particularly with recent advances in supercomputing architectures.

Given the amount of efforts to control performance issues on such large systems where precision is of the utmost priority, and exactness at each stage of the computation makes the difference between a truncated value of zero and an order of 10s of thousands, certain well known scientific computing numeric expressions are stored in the form of a floating point constant.

In a system of clustered machines, bytes representing such a constant are typically indexed in a large register file for the purposes of reloading the value for multiple processes to access. And this is a design choice of necessity with some double values in certain representations, since reciprocal closure of a numerical format can be achieved with well designed fractional mappings of these stored values.

For instance, in [5], one fractional mapping includes $\sqrt{2}$ in the form, and this needs to be stored, as its calculation can be quite expensive[3]. All of this architecture calls for controlled memory accesses, necessitating the creation of memory barriers.

Roy P. Gulla is a Sports/Racebook Writer for the Mesquite Gaming Corp. Mesquite, NV 89027 (Phone:725-208-2328; email: rgulla01@saintmarys.edu)

In these wide bitfield implementations with fractional mappings, it is the reloading and storing of these values which can prove expensive on memory and performance of the processing unit, as the instruction set can become bloated, particularly when multiple precision floating values are being represented.

To worsen matters with stored computing values, with current advances in quantum supercomputing hardware there is an inherent built-in guaranteed error in reading stored bitfields that needs to be accounted for in processing information. In [1], the authors are excited to report that a probability of accurately reading a stored array in a quantum information channel was found to be 1/polylog(n), with n being the rank of the matrix!

Regardless of the processor type or manufacturer, floating point values repeatedly require more machine cycles than their integer type counterparts when performing computations, or even when simply performing loading and storing of the values. And as it is stated immediately above, it is the storage of these values, whether in intermediate or finalized tabulated form, that is a source of major concern where precision and memory constraints are mixed.

In the second portion of this paper, a method will be introduced which takes advantage of the POSIT's minposis mapping, and bypasses the need for the fractional bits of this IEEE float formatting, since POSIT has fractional bits as an optional bitfield anyhow.

II. HARDWARE BACKGROUND

Due to the difference in types of accumulator issues between integer and floating point registers, the following well known but slightly outdated High Performance Super Computing is presented here to simply demonstrate which operations prove more expensive on these IEEE float types. The volatility of certain register operations in this ULTRA SPARC II processor were demonstrated in [1] during a Sun Microsystems Research study conducted with students at a point when multi processor and multi threaded applications were becoming the industry norm.

During this study, students performed computing tasks and recorded the results observed on a performance counter over a significant window of time.

The counter displayed that, in eight of the nine different computing architectures tested the vast majority of compute time was spent on loading and/or storing operations, versus the simple reading and writing from registers.

ISBN: 978-988-14049-1-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

In the ninth architecture, Intel's GNU system displayed the vast majority of time was spent in “misprediction” stalls, or when the processor incorrectly predicted the memory allocator branch taken when a cache miss occurs. Another potential performance hit in these particular parallel architectures can occur when these floating point block load and stores are allowed access to the same memory registers as atomic load and stores.

In order to maximize the usage of compute resources, in light of these performance issues, recently a premium has been placed on allocating a chip on a single atom of computer architecture. With this recent achievement, when a single nanometer of a computing device is used in some arithmetic circuit now, then the order of 10 kilobytes of memory is being consumed. To place a scope on the cost of certain arithmetic circuits on performance, implementations of multiplier circuits in machine learning networks are lucky if they cost 28 nanometers in computer memory width[6]..

To deal with these tradeoffs between precision and limited space, today's architectures can turn to multiple precision computing, in order to combine the space efficiency of single (or even half) precision and the scientific precision of double or higher precision.. One of the issues with these multiple width register sets is that of denormalized values, occurring typically with partial storage approaches to numerical values. A major reason for the denormalized values is when instruction sets are required to bit shift floating point values varying amounts depending on the specified number of exponent bits, which can be varied depending on how modular the system is in it's design[5]. This will then sometimes result in the truncation of the intermediate value to 0.

In fact, a very modular system has been proposed by Lawrence Livermore National Laboratory that proposes a variable number of exponent bits, and new mappings of fraction bits[5]. One of these fractional mappings is mentioned above in light of the current paper's proposal.

III. TRADITIONAL APPROACHES TO ROUNDING OR TRUNCATION.

This new proposed implementation of certain rational numbers avoids the one issue that is common to all of the architectures.mentioned above, storage and retrieval of values. Simply put, the difficulties in the rounding stage of computation are not merely limited to differences in machine precision between a source and a target machine, but rather arise in the storage of intermediate values, particularly in partial summation schemes (as found in accumulators), and the use of these memory operands in the necessary comparisons the machine makes in order to make the final necessary adjustments to the numerical representation of the value. These difficulties arise as multiple comparisons performed on floating point values leads to an additional intermediate (difference) term each time a compare is performed, and ultimately it is the source and not the destination precision settings which take precedence. If the source has chosen truncation as its option for optimizing the floating point unit computation, the effects can be very pronounced.

The use of these stored memory operands was such a hindrance to performance that Intel completely engineered them out of the comparison stage of computation long ago, and so registers(whichever size and type) are the only allowed operands in comparisons for its floating point values today.

The following example using the Intel floating point register st(0) demonstrates the point:

ST(0) REGISTER **val** **st(0) =2.0**	**ASM INSTRUCTION** **fload val** **fsqrt** **fmul ST(0), ST(0)** **fsub val**
	Final ST(0) val: 4.4408921E-016
No. Of FPUs used	**1**

Figure 1 Intel Floating Point register issue with a non-representable number.

The above issue occurred amazingly on a single processor system, with a single FPU. In multiple processor systems the issue could become more pronounced.

The problem being addressed can be demonstrated with the following tabular display of a commonly encountered issue with any binary numbering format, showing how memory constraints can become a huge problem in any system. (It is important to note that by the 10^{th} fractional bit in the following diagram, the represented number is still .0008 off the true value. And after 8 more bits,it overshoots the desired number.

Decimal Value	**.2 (2/10)**
Binary Representation	**.00110011001100110011...(etc)**
No. Of Bits Required	**INFINITE**

Figure 2: In this instance engineers are tempted to incorporate an alternative exponent in the floating point value, thereby mixing varying registers, which may lead to zeroing out of system values

IV. A TABULAR DEMO OF MODEL TO FOLLOW

When accelerators become involved in the intermediate stages of computation, as happens in partial product circuitry, and multiple registers become incorporated in the arithmetic circuits, optimizations can be (and should be) made for certain register types (and their widths). And although Posits share some similar fields as the IEEE floating point type, there is a very distinct difference in their treatment of the fractional bitfield, which is what the last section here hopes to present as one of its most attractive qualities.

ISBN: 978-988-14049-1-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

With a repeating bit pattern as that shown in the above Figure 2, the obvious culprit is a repeating remainder term in the binary fractional division scheme, which itself cannot be transformed into binary formatting. Here the fractional bits are in fact an engineering flaw and a source of both memory and accuracy issues. This particular example is shown as its proposed representation below can be utilized for all numbers of the form $1/10^n$, utilizing the accumulator scheme proposed in [7]. In such a case, the proposal is as follows below:

Decimal Value	**.2 (2/10)**
Alternate Expanded Form, in as much binary as possible->	**1/4 – 1/4(1/5)** **1/4(1 – (1/4(1 – 1/5)** **1/4(1-(1/4(1-(1/4(1-.1/5)** **etc.**
Floating Term	**1-1/5=4/5**

Figure 3:By the fourth iteration in the expanded form in binary, the computation is within .0015 of the actual number , and only 2 bits have been used. After another iteration, we are within .0003 of the number. 16 digits of precision is reached after 18 iterations, using these same two bits.

In the above representation of the decimal value of .2, the first sequence showing the 1/5 term is 1/4 – 1/4(1/5), or 1/4(1 - .2), and the second sequence would be 1/4(1 – (1/4(1 - .2)), and the third 1/4(1-(1/4(1-(1/4(1-.2) etc. Via this route we are within .008 after 3 iterations, and after 6 iterations, within .00001 of the desired number, while avoiding the unrepresentable term. It takes 21 iterations to achieve machine level precision, and in modern supercomputing processors, such mathematical instructions take picoseconds, not nano. Again, only 2 bits from the stack are used. And the problems which follow from the storing and loading of the value in some suitable form, both to and from a table, are completely avoided.

Without loss of generality, the case of 3^i(specifically,its reciprocal field) is presented here. The other fields have similar expanded representations of those numbers not representable in binary form. In each case, the repeated bit term used in the calculation is simply weighted in the implemented circuit. This use of weights is fundamentally important to the last section's treatment of a particular scientific constant.

Decimal Value.	**.333333...(1/3)**
Alternate Expanded form, in as much binary as possible->	**1/4 + 1/12 = 1/4(1+1/3)** **=1/4(1+1/4(1+1/3))** **=1/4(1+1/4(1+1/4(1+1/3)))** **etc.**
Floating Term	**1+1/3 = 4/3**

Figure 4: Another example of an expanded binary form of a rational number, with a non-representable component. The same convergence rate to the actual desired number is seen here as in Figure 3. This approach is useful with expansions of rational numbers, or any reals in something other than an alternating series, with weights, as is commonly encountered in certain engineering and science applications.

V. A COMMON FLOATING POINT CONSTANT WITH A WEIGHTED BIT REPRESENTATION

As the proposed final addition to the proposed alternative formatting designed to deal with rounding issues, a weighted representation of bits is used to compute a common constant typically stored and loaded as a floating point constant, the natural log base. It is presented here as an alternative to the typical storage of the natural log base, which again can be prone to the inherent error issues mentioned in [1].

The iteration scheme followed can be explained as follows:

If at any point in the calculation of the constant, the inclusion of the next bit in the minposis bit set causes the value to overshoot the desired one, simply skip the bit and proceed to the next, and keep reading until the value overshoots, at which point discard the previously read bit.

In the following table, a representation of the natural log base is given to machine level precision, utilizing the extra padded bit field (bits 49, 52, 54) incorporated in the tilepro architecture. This is the suggested mode for any future implementations of the model.

EXAMPLE OF THE ABOVE APPROACH USING THE NATURAL LOG BASE

SET SIGNIFICAND BITS	**2,4,5,7,8,9,10,11,12,17,20,21,22,23, 24,25,26,27,29,30,31,32,33,34,35,3 6,37,38,39,40,42,43,44,45,46,47,49, 52,54*(57,60,65,66,69,70)**
CORRESPONDING BIT WEIGHTS	**2,2,2,2,2,2,2,2,2,2,2,2,4,2,2,2,4,4,2,2 ,2,2,4,2,2,2,8,4,4,4,8,8,8,4,4,4,4,2,2,2,2,1, *(1+,1+,1+,1+,1+,1+,1+)**
+1 SUPERUNITARY BIT§	**1**

§Superunitary as it is used in[3]. Simply put, it is the subset of Reals greater than unity.

Figure 5 The proposed weighted numerical formatting, using the natural log base constant, represented to machine precision of 16 decimal places. No fractional weights are in the list, so the model is implemented with a simple hardware addition circuit.

VI. SUGGESTION FOR FURTHER USE OF THE MODEL

The implementation design here for the natural log base allows for the use of a multiple vector register set, such as that utilized in the AltiVec processor by IBM. And as it has been repeatedly emphasized in this paper, also seems suitable for implementation with the POSIT's minposis subunitary numbers, as described in [3].

ISBN: 978-988-14049-1-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

REFERENCES

[1] Bian, Teng, Daskin, Ammary , Kais, Sabre, Xia, Rongxin "Context Aware Quantum Simulation of a Matrix Stored in Quantum Memory." Quantum Information Processing. 18 10.1007/s11128.019.2469.1
[2] Enbody, Richard J.,j Moore, William, Pellini, Kelley "Performance Monitoring in Advanced Computing Architecture." Sun Microsystems Research Division
[3] Cheng, S. Liang,F. Liang, J., Xiao, F., Wu, B Zhang, G. "Posit Arithmetic Hardware Implementations with the Minimum Cost Divider and Square Root." Electronics 2 October 2020
[4] Gustafson, John. (2015). The End of Error: Unum Computing. 10.1201/9781315161532.
[5] Hittinger, Jeffrey, Lindstrom, Peter, Lloyd, Scott "Universal Coding of the Reals: Alternatives to IEEE Float." February 2018
[6] Johnson, Jeff (2018) "Rethinking Floating Point for Deep Learning."
[7] Koenig, Jack & Biancolin, David & Bachrach, Jonathan & Asanovic, Krste. (2017). "A Hardware Accelerator for Computing an Exact Dot Product." 114-121. 10.1109/ARITH.2017.38.

ISBN: 978-988-14049-1-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)