Accuracy, Cost, and Performance Tradeoffs for Floating-Point Accumulation

Krishna K. Nagar
Department of Computer Science and Engineering
University of South Carolina
Columbia, SC 29208, USA
Email: nagar@email.sc.edu

Jason D. Bakos
Department of Computer Science and Engineering
University of South Carolina
Columbia, SC 29208, USA
Email: jbakos@cse.sc.edu

***Abstract*—Set-wise floating point accumulation is a fundamental operation in scientific computing, but it presents design challenges such as data hazard between the output and input of the deeply pipelined floating point adder and numerical accuracy of results. Streaming reduction architectures on FPGAs generally do not consider the floating point error, which can become a significant factor due to the dynamic nature of reduction architectures and the inherent roundoff error and non-associativity of floating-point addition. In this paper we two frameworks using our existing reduction circuit architecture based on compensated summation for improving accuracy of results. We find that both these implementations provide almost 50% exact results for most of the datasets and relative error is less than that for the reduction circuit. These designs require more than twice the resources and operate at less frequency when compared to the original reduction circuit.**

***Index Terms*—Computer arithmetic; Floating point accumulation; Rounding errors; Numerical accuracy; Compensated summation; FPGA.**

I. Introduction

Floating point summation operation– or accumulation–, $S = \sum_{i=1}^{n} a_i$ is the core of linear algebra operations which comprise the kernel of scientific applications such as fast multipole methods, computational fluid dynamics, networks etc.

In many applications, it is required to accumulate sets of different sizes which also require high numerical accuracy. Achieving high performance for the accumulation operation thus presents two challenges. Firstly, on a parallel computer, the summation can be performed using a parallel "reduction" operation. In this operation, multiple additions are performed concurrently, producing multiple "partial" sums that are maintained separately until they are eventually added into a final result. However, the non-associativity of floating point addition causes the rounding error to be dependent on the run-time behavior.

The second challenge arises from the fact that floating point adders are deeply piplelined. If a new input is delivered to the adder every clock cycle, it cannot provide the current sum before the next value arrives, creating a data hazard. In such a case, the designer cannot use a simple feedback-based accumulator as it may result in addition of values belonging to different sets. In order to accumulate multiple sets of different sizes and deal with data hazard, a special circuit with proper scheduling techniques around floating point adders is required. This requirement leads to control overhead which may require additional resources and may reduce the hardware utilization.

Implementing parallel reduction operations on FPGAs presents some unique opportunities to maintain and improve the accuracy and resolve the issue of data hazard. A rich set of tradeoffs can be made between error bound and resource usage through the design of customized floating point units. Also, very fine-grained control logic can be used to reduce the overheads of data dependent behavior.

Several methods have been developed and studied in order to improve the accuracy of summation. Some of these techniques have also been implemented on FPGA based coprocessors[16], [17], [18], [19]. Also, there have been several efforts to implement streaming set-wise reduction on FPGAs[11], [12], [13], [14], [15].

The motivation of this research arises from the observation that previous works address the problem of improving accuracy and streaming set-wise reduction of floating point values separately. In this paper, we present a high performance general purpose reduction circuit which solves with the problem of floating point accumulation. This reduction circuit achieves nearly 100% utilization regardless of the structure of the input data. We also examine methods to leverage custom hardware in order to place tighter error bounds on the accumulation operation. We incorporate these methods in our reduction circuit.

II. (In)Accurate Floating Point Addition

Methods, in which large floating point datasets are used, may deliver inaccurate results due to different sources of errors. Stability of numerical algorithms is an important topic of research in the field of numerical analysis and much of the focus has been given to the accuracy of floating point operations [1], [2].

Rounding errors are unavoidable due to prevalence of finite precision floating point arithmetic[3]. Rounding errors can be introduced in two ways- shift and carry. The error due to shift

978-1-4799-2079-2/13/$31.00

operation of smaller operand during addition causes shifting error. A nonzero carry which results the significand width to be more than the allowed bits causes carrying error.

Error during a floating point addition can be calculated using additional floating point operations using algorithm 1 which is known as Fast2Sum[6]. The error can be incorporated into intermediate and the final results[10]. Such methods are known as compensated summations.

Kahan's compensated summation algorithm which calculates and applies the correction in each iteration for recursive summation[4]. In this algorithm, the error term *e* - the approximation to the rounding error, is subtracted from the next input value in subsequent iteration. Another step can be added to this algorithm in which *S* + *e* can be calculated at the end of the loop [5]. It has been shown that this algorithm improves the error bound and gives almost ideal result. Equation 1 gives the relative error bound for Kahan's compensation algorithm which is independent of *n* if $n\epsilon < 1$.

$$\frac{|E_n|}{|S_n|} \leqslant (2\epsilon + O(n\epsilon^2)) \frac{\sum_{i=1}^{n} |a_i|}{|\sum_{i=1}^{n} a_i|} \tag{1}$$

The condition number in the above equation is

$$\kappa = \frac{\sum_{i=1}^{n} |a_i|}{|\sum_{i=1}^{n} a_i|} \tag{2}$$

$|E_n|$ is the difference between the exact summation calculated with infinite precision, $|S_n|$ and the recursive summation in floating point precision, S_n. This error bound considers the error introduced at each iteration step of the recursive addition of terms *a*; and it is dependent on the number of terms, *n*, but independent of the order in which they are added. Here, ϵ represents the machine precision.

Algorithm 1 Fast2Sum Algorithm

1: $Input(a, b)$
2: **if** $|a|<|b|$ **then**
3: $\quad swap(a, b)$
4: **end if**
5: $x = a + b$
6: $b_t = x - a$
7: $e = b - b_t$
8: $Return(x, e)$

Another version of compensated summation has been developed where the error terms calculated after each addition, using Fast2Sum algorithm, are accumulated and the correction is applied at the end of the summation. Equation 3 shows the relative error bound for this method.

$$\frac{|E_n|}{|S_n|} \leqslant (2\epsilon + O(n^2\epsilon^2)) \frac{\sum_{i=1}^{n} |a_i|}{|\sum_{i=1}^{n} a_i|} \tag{3}$$

It must be noted that Fast2Sum algorithm requires $|a| \geqslant |b|$. This essentially creates a branch in software implementations of these algorithms but when implementing these in hardware, the swap operation in floating point adder eliminates the need

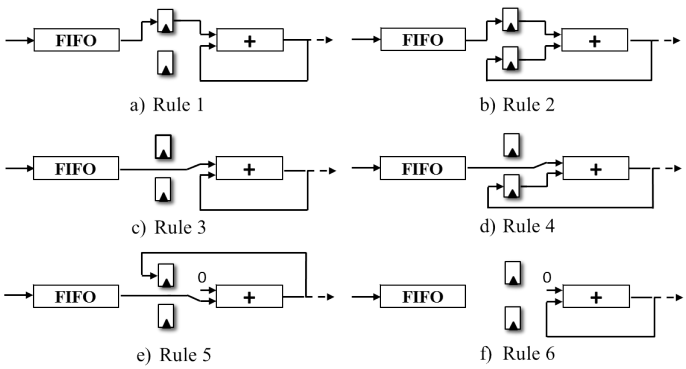

Fig. 1: Reduction Circuit Rules

of this check and thus accurate summation can be calculated with three additional steps. Also, these methods may have large relative error when the condition number is high but overall the results are more accurate then recursive summation.

Several other variations compensated summation techniques have been developed but require significantly more number of floating point operations[7], [8], [9].

It can be observed that in compensated summation methods, additional steps are required to recover the error encountered during the alignment operation. But these do not require a-priori knowledge of the data hence they are more suitable and less expensive for hardware implementation.

III. Dataset Reduction

In this section we present an approach to overcome the issue of data hazard in set-wise accumulation of floating point values. This design was originally published in [21].

The reduction circuit has been designed by adding control logic- comparators, counters, and buffers- around a standard deeply pipelined double precision adder in order to form a dynamically scheduled accumulator.

The reduction circuit consists of a set of data paths that allow input values and the adder output to be delivered into the adder or buffered based on their corresponding accumulation set ID and the state of the system. Data paths are established by the control unit according to six rules as shown in Figure 1

This architectures consists of 4 buffers which store the incoming values and the partial sums. The input values are supplied to the reduction circuit through a FIFO. In case, all the buffers are occupied and rules 1 to 5 cannot be applied, input value is not read from the FIFO and rule 6 is applied.

IV. Error Compensation in Hardware

In this section, we present a set-wise floating point accumulation framework for FPGAs which not only reduces multiple streaming sets efficiently but also improves the accuracy of the results. The objective of this is to evaluate various design trade-offs such as resource usage and working frequency for different methods. Our goal is to achieve high throughput while maintaining high accuracy and keeping the resource requirement low. The reduction circuit architecture described

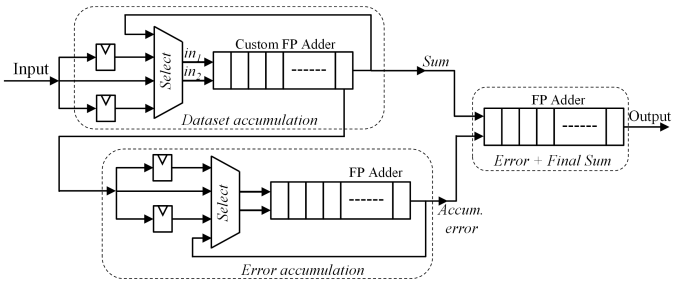

Fig. 2: Accumulated Error Compensation

previously serves as the foundation for the error correction frameworks.

We have designed a custom double precision adder based FloPoCo[20] adder design which outputs the error encountered during addition along with the sum. This adder not only takes into consideration the shifted out bits but also the error due to rounding. The custom adder essentially reduces one floating point operation from the algorithms and hence reduces the overall latency of the algorithms. As listed in Table I, the resources required for the custom adder are almost twice and the operating frequency suffers 29% performance penalty.

In the following sections, we discuss two designs based on compensated summation to improve the accuracy.

A. Accumulated Error Compensation

Using the custom adder, we have implemented compensated summation algorithm where the error generated from the custom floating point adder is accumulated. We call this design *accumulatederror compensation(AEC)*. In this design, two reduction circuits are required- first reduction circuit sums up the dataset values and output the errors. We call this value reduction circuit (VRC). The second reduction circuit, called Error Reduction Circuit (ERC), accumulates the errors being generated from VRC. Once the set is completely reduced in VRC, the accumulated error is added to the final sum using another floating point adder. The difference between ERC and VRC is that the values belonging to a particular set are delivered contiguously to VRC but ERC may receive the error values non-contiguously. This can potentially lead to a false reduction in ERC. To prevent this situation, ERC checks for the valid_out signal from VRC to check if the set has been reduced in VRC. Figure 2 shows an overview of AEC.

TABLE I: Comparison of Designs

Design	Slice Register	Change	Frequency	%Change
FP Adder	1130		330 MHz	
Custom Adder	2310	2.04X	235 MhZ	-28.7%
Reduction Circuit	1873		188 MHz	
AEC	4743	2.53X	176 MHz	-6.3%
AECSA	7938	4.23X	135 MHz	-29.2%

B. Adaptive Error Compensation in Subsequent Addition

In another compensated summation approach, have implemented a method where the error can be added to value in subsequent addition if available. We refer to this approach as *adaptive error compensation in subsequent addition (AECSA)*. In this design, two reduction circuits are required. In the first reduction circuit, VRC, the two adders are required. Before the first adder, we need to compare the input values and supply the value with greater exponent to the first adder. The first adder calculates the difference between the input and the error previously generated. Custom adder connected sequentially to the output of the first adder calculates the sum of inputs and the associated error. Another adder is required to add the final result and the last error value. Since we allow multiple accumulations from the same set simultaneously, there may arrive a case where we may not be able to add the error. In such a case we accumulate such errors using ERC. Partially accumulated errors can also be supplied from ERC to VRC. If the error term comes from ERC, it is invalidated in ERC and is not considered for accumulation. The rule to be applied in ERC depends on whether the corresponding error term has been invalidated or not. Thus, supply of error term gets priority over accumulation in ERC. This approach has been depicted in Figure 3.

As listed in Table I, AEC requires 2.5 times more resources and operates at 6.3% less frequency as compared to the original reduction circuit while AECSA requires 4.2 times more resources and operates at 29.2% less frequency on Xilinx Virtex 5 LX330 FPGA. We attribute the increase in the resource usage and reduction in frequency to the custom adder, ERC and the additional control logic to implement ERC. Also, in AECSA, it is due to the supply of error from ERC.

V. Accuracy of Designs

In order to test the circuits for accuracy, we generated random datasets with some constraints and divided those datasets into sets of size 100. We simulated the designs and compared the results against the results from original reduction circuit without error compensation. In order to calculate the relative error, we used the GNU MPFR[22] library for recursive summation with the precision set to 2048. This is treated as sum with infinite precision, $|S_n|$. We report the dataset attributes, percentage of exact results and the relative error in Table II. The shift amount during addition depends on the difference between exponents hence we generated datasets with different range of exponents. Also, the relative error is large if the condition number, κ, is greater than 1 i.e. the datasets contain both positive and negative number and $\sum_{i=1}^{n} |a_i| > |\sum_{i=1}^{n} a_i|$. For $\kappa = \infty$, we report absolute error $|S_n| - S_n$.

It is evident that the relative errors from AEC and AECSA are comparable in all the cases while the relative error is high without error compensation. The relative error becomes more prominent if the condition number of dataset is high, yet it is significantly less for AEC and AECSA than that for the design without error compensation. Also, these designs provide almost 50% exact results ($|S_n| - S_n = 0.00$) in most cases. Thus, our designs with error compensation achieve better accuracy in set-wise summation.

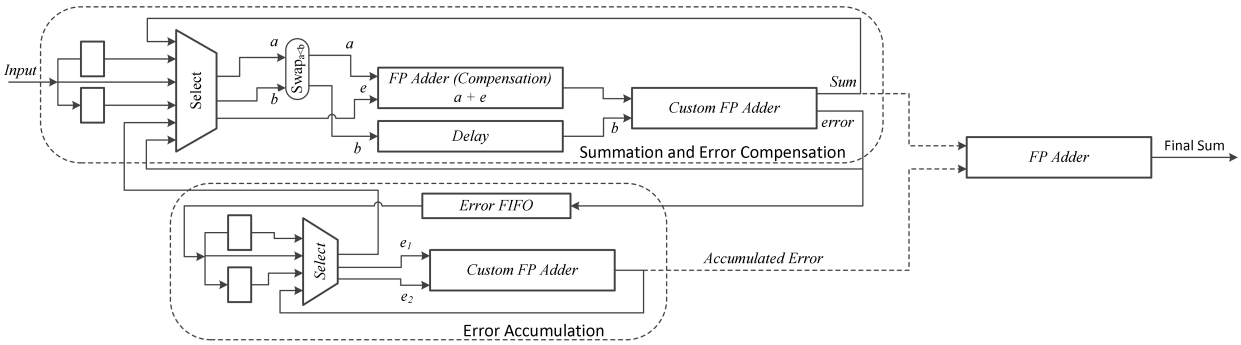

Fig. 3: Adaptive Error Compensation in Subsequent Addition

TABLE II: Relative Errors for Designs

Property			Reduction Circuit		AEC		AECSA	
Dataset Range	Exponent	κ	Exact	Rel. Error	Exact	Rel. Error	Exact	Rel. Error
0 to 1	exp = 1	1.0	0.00%	3.14E-16	50.32%	1.68E-16	49.42%	1.74E-16
2 to 4	exp = 1	1.0	0.00%	3.87E-16	48.90%	2.01E-16	49.03%	2.02E-16
2 to 32	$1 \leqslant \exp < 5$	1.0	0.00%	2.89E-16	51.87%	1.59E-16	49.94%	3.00E-16
2 to 2048	$1 \leqslant \exp < 11$	1.0	0.00%	3.46E-16	51.10%	1.73E-16	51.35%	2.93E-16
2 to 2^{50}	$1 \leqslant \exp < 50$	1.0	0.00%	3.99E-16	54.45%	2.21E-16	51.74%	4.01E-16
-1 to 1	$\exp < 1$	> 1.0	0.00%	1.34E-13	41.16%	1.85E-14	31.35%	2.80E-14
-16 to 16	$1 \leqslant \exp < 4$	∞	0.00%	2.84E-14	58.58%	2.66E-15	32.97%	1.69E-14

VI. Conclusion

In this document, we have presented two designs for accurate set-wise floating point summation based on our reduction circuit and compensated summation algorithms. On the basis of our results, we can conclude that using compensated summation in hardware provides more accurate results but requires significantly more resources and operate at less frequency due to additional control logic.

References

[1] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2002.

[2] Ivo Babuska. Numerical stability in mathematical analysis. In *IFIP Congress (1)*, pages 11–23, 1968.

[3] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.*, 23(1):5–48, March 1991.

[4] W. Kahan. Pracniques: further remarks on reducing truncation errors. *Commun. ACM*, 8(1):40–, January 1965.

[5] W. Kahan. *A Survey of Error Analysis*. Defense Technical Information Center, 1971.

[6] T.J. Dekker. A floating-point technique for extending the available precision. *Numerische Mathematik*, 18:224–242, 1971/72.

[7] Siegfried M. Rump, Takeshi Ogita, and Shinichi Oishi. Accurate floating-point summation. Technical report, University of California, Berkeley, 2005.

[8] Peter Kornerup, Vincent Lefevre, Nicolas Louvet, and Jean-Michel Muller. On the computation of correctly rounded sums. *IEEE Transactions on Computers*, 61(3):289–298, 2012.

[9] Jonathan Richard Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete and Computational Geometry*, 18:305–363, 1996.

[10] James Gregory. A comparison of floating point summation methods. *Commun. ACM*, 15(9):838–, September 1972.

[11] Ling Zhuo, Gerald R. Morris, and Viktor K. Prasanna. High-performance reduction circuits using deeply pipelined operators on fpgas. *IEEE Trans. Parallel Distrib. Syst.*, 18(10):1377–1392, 2007.

[12] Miaoqing Huang and David Andrews. Modular design of fully pipelined reduction circuits on fpgas. *IEEE Transactions on Parallel and Distributed Systems*, 99(PrePrints):1, 2012.

[13] Krishna K. Nagar and Jason D. Bakos. A High-Performance Double Precision Accumulator. In *Proceedings of the 2009 International Conference on Field-Programmable Technology (FPT'09)*, pages 255–262, 2009, Sydney, NSW, Australia.

[14] M. E. T. Gerards, J. Kuper, A. B. J. Kokkeler, and E. Molenkamp. Streaming reduction circuit. In *Proceedings of the 12th EUROMICRO Conference on Digital System Design, Architectures, Methods and Tools, Patras, Greece*, pages 287–292, Los Alamitos, August 2009. IEEE Computer Society.

[15] Michael deLorimier and André DeHon. Floating-point sparse matrix-vector multiply for fpgas. In *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, FPGA '05, pages 75–85, New York, NY, USA, 2005. ACM.

[16] Nachiket Kapre. Optimistic parallelization of floating-point accumulation. In *18th Symposium on Computer Arithmetic*, pages 205–213. IEEE, 2007.

[17] Chuan He, Guan Qin, Mi Lu, and Wei Zhao. Group-alignment based accurate floating-point summation on fpgas. In *ERSA'06*, pages 136–142, 2006.

[18] Manouk V. Manoukian and George A. Constantinides. Accurate floating point arithmetic through hardware error-free transformations. In *Proceedings of the 7th international conference on Reconfigurable computing: architectures, tools and applications*, ARC'11, pages 94–101, Berlin, Heidelberg, 2011. Springer-Verlag.

[19] Edin Kadric, Paul Gurniak, and Andre DeHon. Accurate parallel floating-point accumulation. In *Proceedings of the 2013 IEEE 21st Symposium on Computer Arithmetic*, ARITH '13, Washington, DC, USA, 2013. IEEE Computer Society.

[20] Florent de Dinechin and Bogdan Pasca. Designing custom arithmetic data paths with FloPoCo. In *IEEE Design and Test of Computers*, 28(4):18–27, 2011

[21] Krishna K. Nagar and Jason D. Bakos A Sparse Matrix Personality for the Convey HC-1. In *Proceedings of the 19th Annual IEEE International Symposium on Field Programmable Custom Computing Machines*(FCCM'11), Salt Lake City, UT, USA. May 2011.

[22] The GNU MPFR Library. http://www.mpfr.org, Aug 2013.