Group-Alignment based Accurate Floating-Point Summation on FPGAs

Chuan He, Guan Qin, Mi Lu, and Wei Zhao
Texas A&M University, College Station, TX 77843
chuanhe@ee.tamu.edu, guan.qin@tamu.edu, mlu@ee.tamu.edu, w-zhao@tamu.edu

Abstract

Floating-point summation is one of the most important operations in scientific/numerical computing applications and also a basic subroutine (SUM) in BLAS (Basic Linear Algebra Subprograms) library. However, standard floating-point arithmetic based summation algorithms may not always result in accurate solutions because of possible catastrophic cancellations. To make the situation worse, the sequence of consecutive additions will affect the final result, which makes it impossible to produce a unique solution for the same input dataset on different computer platforms with different software compilers.

The emergence of high-density reconfigurable hardware devices gives us an option to customize high-performance arithmetic units for our specific computing problems. In this paper, we design an FPGA-based hardware algorithm for accurate floating-point summations using group alignment technique. The corresponding full-pipelined summation unit is proven to provide similar or even better numerical errors than standard floating-point arithmetic. Moreover, it consumes much less RC resources as well as pipelining stages than existent designs, but achieves the optimal working speed at one summation per clock cycle with only moderate start-up latency. This new technique can also be used to accelerate executions of other linear algebra subroutines on FPGAs and result in much more efficient and compact implementations without negative impact on computational performance or numerical accuracy.

1. Introduction

Floating-point arithmetic is preferable to fixed-point arithmetic in computationally intensive scientific /numerical applications because of its ability to scale the exponent for a wide range of real numbers. In 1994, Fagin et al [1] first testified the feasibility of implementing single-precision floating-point arithmetic units on FPGAs, though their performance was uncompetitive to contemporary commodity computers. Since then, as FPGA devices continue to grow in density and speed, their attainable floating-point performance has been increasing significantly faster than commercial CPUs. Architectural changes like implanting of hardwired multipliers and RAM blocks further accelerate this trend. In [2], the authors predicted that FPGA devices would yield three to eight times more peak floating-point performance than commodity CPUs by 2009.

With commercial or open-source parameterized floating-point libraries [3] [4], implementing floating-point computations on FPGAs is straightforward and convenient. However, this approach requires much more reconfigurable hardware resources than their fixed-point counterparts and lead to excessive computation latency. For example, constructing a fully pipelined single-precision floating-point adder on Xilinx Virtex-II Pro devices consumes over 500 logic slices with 16 pipeline stages [5], and a double-precision unit with similar performance costs nearly 1000 logic slices with 19 pipeline stages. For special scientific/engineering problems on hand, it is always possible, and indeed desirable, to adopt customized floating-point formats by making tradeoffs among accuracy, area, throughput, and latency [6] [7]. Efforts were also made to investigate the possibility of replacing floating-point operands and operations by fixed-point arithmetic thoroughly [8] [9]. However, most of these works only demonstrated the feasibility by numerical evidences without rigorous mathematical proofs, so are unconvincing for religious scientists.

Instead of directly enhancing standard floating-point arithmetic units, in this work, we choose to improve the performance and accuracy of numerical algorithms involving a bunch of fundamental floating-point operations. Particularly, we designed a group-alignment based hardware algorithm to accumulate a set of floating-point operands accurately and efficiently. Here, we interpret accuracy as similar or even better relative/absolute rounding errors comparing with standard floating-point arithmetic. Hardware efficiency means that the resulting summation unit consumes comparable or less RC resources on FPGAs than existent designs [13] [14], but can always achieve its optimal working speed at one summation per clock cycle with only moderate start-up latency. This technique can also be applied to other linear algebra routines like vector norm, dot product,

matrix-vector multiplication, and matrix-matrix multiplication to efficiently decrease RC resource occupations and reduce pipeline stages without negative impact on computational performance or numerical accuracy.

The rest of this paper is organized as follow: In Section 2, after a short review on the underlying floating-point summation problem, we introduce the formal upper bound of numerical rounding errors associated with the naïve sequential accumulation algorithm. The new group-alignment based summation algorithm is presented and analyzed in Section 3. Rigorous proof and numerical experiments are provided to prove that this new approach can provide similar or even better numerical rounding errors than standard methods. In Section 4, this new hardware algorithm is implemented on a XUPV2P Virtex-II Pro evaluation board to show its performance superiority over previous designs. Finally, conclusions and a discussion of future research directions are presented in Section 5.

2. Problem Statement and Related Work

2.1 Accurate floating-point summation

Floating-point summation is such an important operation in numerical computations that some computer scientists even suggested to import it into general-purpose CPUs as "the fifth floating-point arithmetic" [10]. Unlike other fundamental floating-point arithmetic, summation is not well-conditioned because of so-called "catastrophic cancellations" [11], i.e. small relative errors hidden in inputs might result in much larger relative error in output. To make the situation worse, the sequence of consecutive additions will also affect the final sum, which makes it impossible to produce a unique solution for the same input data set on different computer platforms with different programming languages or software compilers [12]. On the bright side, this un-uniqueness relieves us from strictly obeying the IEEE standard in our implementation but choose the exact solution as the only accuracy criterion.

Given a set of n floating-point numbers $s_1, s_2, \cdots, s_n$ with small relative errors ε_M, our goal is to design a numerical algorithm as well as its FPGA-based hardware implementation to compute the summation $S = \sum_{i=1}^{n} s_i$ accurately and efficiently. Specifically, we assume that the summation unit has only one input port to feed in summands as well as one output port to send out the final result. This assumption is consistent with the memory-bandwidth-bounded property of the summation method because there is only one floating-point addition for each summands. Following criterions are used to compare the performance of our design with others:

a) Efficiency. The summation unit consumes comparable RC resources and can work at similar or even higher clock rate as a standard fully-pipelined floating-point adder.
b) Throughput. The fully-pipelined summation unit should work at its highest sustained speed accepting a new summand at every clock cycle without excessive pipelining stalls.
c) Latency. The summation result should be available with only moderate latency after the last summand entering the arithmetic unit.
d) Accuracy. The summation result should have similar or better relative and absolute errors than the result produced by the standard sequential accumulation algorithm with IEEE-754 compliant floating-point arithmetic.

At first glance, a simple accumulator using one floating-point adder with registered output connecting to one input port can have the job done perfectly. However, unlike a single-cycle fixed-point accumulator whose output can be feed back immediately as intermediate results for consecutive summations, the data dependency associated with the fully-pipelined floating-point adder will severely slow down the computational throughput of the corresponding accumulator. This deficiency obviously doesn't meet the requirement we just mentioned in criterion (b). Another extreme is to accumulate n floating-point numbers using a binary tree based reduction circuit with $(n-1)$ adders and $\log_2 n$ tree levels. If pipelining technique is properly applied, this approach can accept a new set of inputs and produce a new output at each clock cycle, so obviously are too luxury for our single input/output unit and won't result in an efficient implementation as we required in criterion (a).

2.2 Related works

One possible way to address the data dependency problem caused by pipelined addition is to introduce a "schedule" circuit and carefully-designed input/intermediate-sum buffers besides a conventional fully-pipeline adder [13]. This approach treats all intermediate-sums the same way as input summands, so the original $(n-1)$ summations with n summands are now converted to $(n-1)$ additions with $(2n-2)$ addends, which ideally can be finished by an adder within $(n-1)$ clock cycles. However, limited by the only external input port of the summation unit, the adder has to rely on its own output to feed back operands, which may not always arrive on time because of the adder's deeply-pipelined internal stages. The main task of the scheduler is to monitor the internal dataflow of the pipelined adder and do its best to feed the adder's two input ports with operands so that the summation task can be finished as soon as possible. Because considerable idle cycles are always inserted at the beginning and the end of the process of summation, latency may become a

potential problem for this approach, especially when n is a small number. Moreover, the scheduler requires considerable register resources to buffer inputs and intermediate-sums, which may not be feasible on FPGAs without internal SRAM blocks.
In [14], the authors proposed a technique named delayed addition to remove carry propagation from the critical paths of pipelined integer and floating-point accumulations. The corresponding FPGA-based implementation meets our requirements for high-throughput and low-latency. However, because considerable hardware resources are consumed to construct Wallace-tree based 3-2 compressors as well as overflow detection/handling circuit, this summation unit is four times more expensive than what we expected in criterion (a). Furthermore, unlike the design in [13], where the accuracy is guaranteed by the standard floating-point adder, the authors only demonstrated the feasibility of this approach by simple numerical tests without rigorous proof, so its correctness and accuracy is still questionable.

3. Group-alignment Based Accurate Floating-point Summation

Instead of following existing commonly-used high-accuracy summation algorithms [15], which are all well-tuned for commercial CPUs so may not ideal for FPGA-based RC platforms, we propose in this section a group-alignment based summation algorithm using customized floating-point/fixed-point hybrid arithmetic. Formal error analysis is presented here to prove that our hardware-based algorithm can always result in similar or even better average/worst-case absolute and relative errors than standard floating-point arithmetic. Numerical experiments are also performed to show the accuracy and scalability of this new approach.

3.1 Errors of the sequential accumulation algorithm

We first analyze error properties of the naïve sequential accumulation algorithm. Assuming the rounding scheme of a machine with standard floating-point arithmetic is round to nearest-even, we have the following accuracy bounds for fundamental floating-point arithmetic:

Lemma1: Let ε_M be the unit rounding-off on a machine with standard floating-point arithmetic ($\varepsilon_M = 2^{-24}$ for single-precision arithmetic, or $\varepsilon_M = 2^{-53}$ for double-precision arithmetic). Then the absolute error and relative error for fundamental floating-point operations (op can be either $+, -, \times, or \div$) may be represented as:

$$\left| fl(x\ op\ y) - (x\ op\ y) \right| \le \left| x\ op\ y \right| \cdot \varepsilon_M \tag{3.1}$$

$$\left| fl(x\ op\ y) - (x\ op\ y) \right| / \left| x\ op\ y \right| \le \varepsilon_M \tag{3.2}$$

Keeping this result in mind, we can easily prove with mathematical induction that computing the summation of $S = s_1 + s_2 + \cdots + s_m$ using the naïve sequential accumulation algorithm can be represented as:

$$\begin{aligned} fl(S) &= fl(fl(fl(s_1 + s_2) + s_3) + \cdots + s_m) \\ &= s_1 \cdot (1+\eta_1) + s_2 \cdot (1+\eta_2) + \cdots + s_m \cdot (1+\eta_m) \end{aligned} \tag{3.3}$$

$$\text{where } |\eta_i| \le (n-i+1) \cdot \varepsilon_M \tag{3.4}$$

So the worst-case absolute error introduced by this method is:

$$\begin{aligned} |e| &= |fl(S) - S| = |s_1 \cdot \eta_1 + s_2 \cdot \eta_2 + \cdots s_m \cdot \eta_m| \\ &\le \left(|s_1| + |s_2| + \cdots + |s_m| \right) \cdot (m-1)\varepsilon_M \\ &\le \max\left\{ |s_1|, |s_2|, \cdots, |s_m| \right\} \cdot m \cdot (m-1)\varepsilon_M \end{aligned} \tag{3.5}$$

And the corresponding relative error is:

$$\frac{|e|}{|S|} \le \frac{|s_1| + |s_2| + \cdots + |s_m|}{|s_1 + s_2 + \cdots + s_m|} \cdot (m-1)\varepsilon_M \tag{3.6}$$

where $\frac{|s_1| + |s_2| + \cdots + |s_m|}{|s_1 + s_2 + \cdots + s_m|}$ is the condition number of floating-point summations.

The "less than or equal to" signs in expression (3.5) and (3.6) imply that the error bound we got here is tight. Putting it another way, we can always encounter the worst case if we got bad luck in choosing the summation sequence or the input data set. We can also note from Equation (3.3) and (3.4) that utilizing the standard floating-point arithmetic doesn't result in a unique solution for different summation sequences.

3.2 Group-alignment based floating-point summations

The un-uniqueness of floating-point summations relief us from strictly complying with the IEEE standard but choose the exact solution as the only accuracy criterion. Here, we propose the group-alignment based floating-point summation algorithm as follow:

Input: A set of n floating-point numbers $s_1, s_2, \ldots, s_n$, which are partitioned into groups with m summands
Output: The summation result S
For k=1 to n/m

1. *Split each summand s_i in this group into two parts as Fraction F_i and Exponent E_i, Restore the hiding bit in F_i to form the mantissa M_i*
2. *Find the largest Exponent E_{max} within E_i*
3. *Calculate every (E_{max}-E_i)*
4. *Shift M_i to right by (E_{max}-E_i) bits, round the shifted mantissas to nearest-even if necessary.*
5. *Sum up all shifted M_i*
6. *Feed rounded M_{sum} and E_{max} to a simplified floating-point accumulator*

End For

Algorithm 1. Group-alignment based FP summation

The basic idea of this group-alignment based summation algorithm is pretty straightforward: To accumulate a group of m floating-point summands, instead of (m-1) standard floating-point additions, where all exponent comparisons and mantissa shifting are scattered, we collect them together to form a unique exponent comparator in Step 2 and a barrel shifter in Step 4. Now, all summands in the same group are aligned into consistent fixed-point format, so can be summed up with simple fixed-point accumulator in Step 5. Figure 1 depicts all steps of this floating-point summation algorithm. This hardware-based method works well for floating-point accumulators with single input port as well as multiple input ports. We will revisit details of our Xilinx Virtex-II Pro based implementation in Section 4.

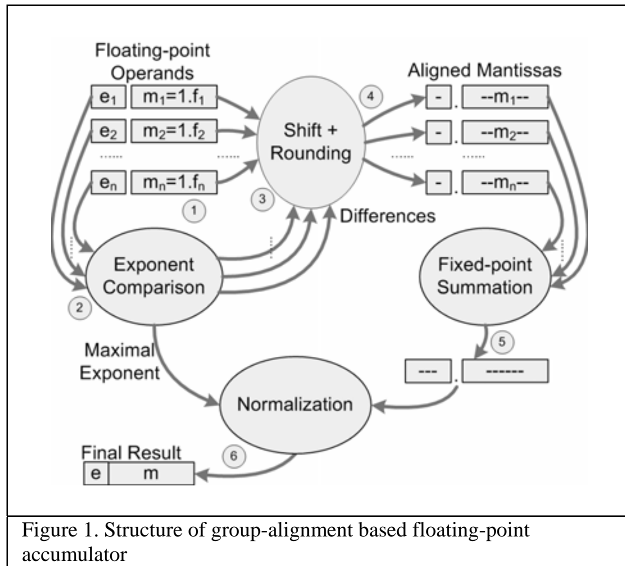

Figure 1. Structure of group-alignment based floating-point accumulator

3.3 Error analysis and comparison

We predict intuitively that the group-alignment technique makes floating-point summations more accurate than standard arithmetic because almost all rounding errors appearing in partial-sums within a group are eliminated except the last one in Step 6. To prove the correctness of this declaration, let's consider the group-alignment based summation with m summands. Because this algorithm utilizes the same monotone rounding scheme (round-to-nearest, towards-zero or away-from-zero) as standard floating-point arithmetic, averaging will have the same effect on rounding error propagations, so only the worst cases are analyzed here.

Lemma 2: The group-alignment based summation algorithm can always result in a unique solution for the same floating-point summand group.

Proof: In Step 4 of the algorithm, all summands within a group are aligned to the one with the largest exponent. Also because a fixed-point accumulator is used in Step 5, which introduces no rounding errors during computations, unique solutions can always be achieved regardless of the sequence of those summands are accumulated.

Let ε'_M be the unit rounding-off on a machine with the new summation unit. Its value now equals to half of the minimal quantity represented by the mantissa accumulator, which is set to be the same as standard floating-point arithmetic temporarily, i.e., 23 fraction bits for single-precision inputs or 52 bits for double-precision. Numerical error analyses show us the following result:

Lemma 3: The group-alignment based summation algorithm achieves similar or even better worst case relative and absolute errors than the sequential accumulation algorithm with standard floating-point arithmetic, depending on the choice of ε'_M.

Proof: After aligning all mantissas to the one with maximal exponent, the absolute rounding error for each right-shifted floating-point summand except the largest one (which introduces no rounding error assuming the number itself is exact.) can be represented as:

$$|e_i| \le \max\{|s_1|,|s_2|,\cdots,|s_m|\}\cdot\varepsilon'_M \tag{3.7}$$

So the maximal total absolute error after fixed-point accumulation is:

$$|e| \le (m-1)\cdot\max\{|s_1|,|s_2|,\cdots,|s_m|\}\cdot\varepsilon'_M \tag{3.8}$$

The corresponding maximal relative error is:

$$\begin{aligned}\frac{|e|}{|S|} &\le \frac{\max\{|s_1|,|s_2|,\cdots,|s_m|\}}{|s_1+s_2+\cdots+s_m|}\cdot(m-1)\cdot\varepsilon'_M \\ &\le \frac{|s_1|+|s_2|+\cdots+|s_m|}{|s_1+s_2+\cdots+s_m|}\cdot(m-1)\cdot\varepsilon'_M\end{aligned} \tag{3.9}$$

Comparing Expression (3.9) with (3.6), we conclude that Lemma 3 holds.

Observing Expression (3.9), we note another efficient method on FPGA-based RC platform to further improve the error bound: It's very easy to decrease ε'_M by using more fraction bits to represent those aligned summands as well as corresponding fixed-point summations. On the contrary, this approach is very painful on general-purpose computers because people have to simulate floating-point operations with double-extended or double-double operands by software code [16].

3.4 Numerical experiments

A MATLAB program is used here to demonstrate error properties of our new summation algorithm. We first create one million groups of single-precision floating-point operands as input data sets, each of which contains ten uniformly-distributed random summands ranging from -0.5 to 0.5. The group-alignment based summations are executed for each input data set and the worst case absolute and relative errors are recorded and compared

with results produced by the sequential accumulation algorithm with standard single-precision arithmetic.

Table 1. Errors for the new summation algorithm

Fraction bits	Maximal/Average absolute error (Condition number)	Maximal relative error (Condition number)
23	5.14e-7/9.03e-8(73.5)	0.143 (4.13e+6)
24	2.68e-7/4.94e-8(1.58)	0.0857 (4.13e+6)
25	1.08e-7/1.72e-8(1.98)	0.0857 (4.13e+6)
26	5.22e-8/2.77e-9(4.51)	0.0182(5.97e+6)
27	1.86e-8/1.78e-9(2.49)	6.85e-4(1.71e+5)
28	7.45e-9/5.06e-10(2.88)	1.08e-4(1.43e+5)
29	3.73e-9/1.36e-10(2.06)	7.65e-5(2.35e+5)
32	2.33e-10/2.24e-12(16.72)	2.13e-6(3.25e+4)
Single-precision	5.29e-7/4.02e-8(1.45)	0.0857(4.13e+6)

Table 1 lists the recorded maximal/average absolute errors and maximal relative errors for the new summation algorithm with different fraction bits. Condition numbers for summations of corresponding input data set are also calculated to show their impacts on final results. Using double-precision arithmetic results as reference, important observations are listed as follow:

- The same conclusion as lemma 3 regarding the worst case/average absolute errors can be drawn from the first column in Table 1. Furthermore, these errors decrease proportionally to the number of fraction bits we adopted. This observation agrees well with the error expression (3.8).
- Numerical stability theory [17] told us that:

$$|Output\ relative\ error| \leq Condition\ number \times |Input\ relative\ error| \tag{3.10}$$

  Catastrophic cancellations happened in those listed worst cases in column two because condition numbers there are all very large. Initial relative errors hidden in input data set are magnified dramatically so that error digits are now much closer to the most significant digit in solutions. Here, we cannot observe a clear relation between errors and the number of fraction bits as in column 1. The reason is that these bad-conditioned cases are still far from the worst ones so that most details are hidden by the “less than or equal to” sign in Expression (3.9). We can easily cook up a floating-point data set with all digits of their summations contaminated.
- Comparatively, condition numbers listed in column one are all moderate. An intuitive explanation to this coincidence is that when condition number of summation is small, most significant operands are with the same sign so that the absolute error bound in expression (3.8) is tight. However when condition number of summation is large, results are much smaller than the largest operand so that those significant operands tend to have opposite signs and cancel others.
- Although the new summation algorithm has better worst-case error bounds, it doesn’t mean that this approach can always provide better results for every input data set. For example, the average absolute error for our algorithm with 23 fraction bits is about two times worse than the conventional single-precision approach. However, it is a little unfair to compare these two cases because a typical implementation of single-precision floating-point adder generally has three more guarding bits.

4. FPGA Implementation

Using the simplest single-precision floating-point accumulator as example, we introduce implementation details of the group-alignment based summation algorithm on FPGAs. Extending this design to double-precision or extended-precision is easy and straightforward with nearly the same performance if proper pipelining technique was applied. An entry-level Virtex II Pro evaluation board [18] is used as the target platform, and the software development environments are Xilinx ISE 7.1i and ModelSim 6.0 se. Figure 2 shows the hardware structure of the summation unit.

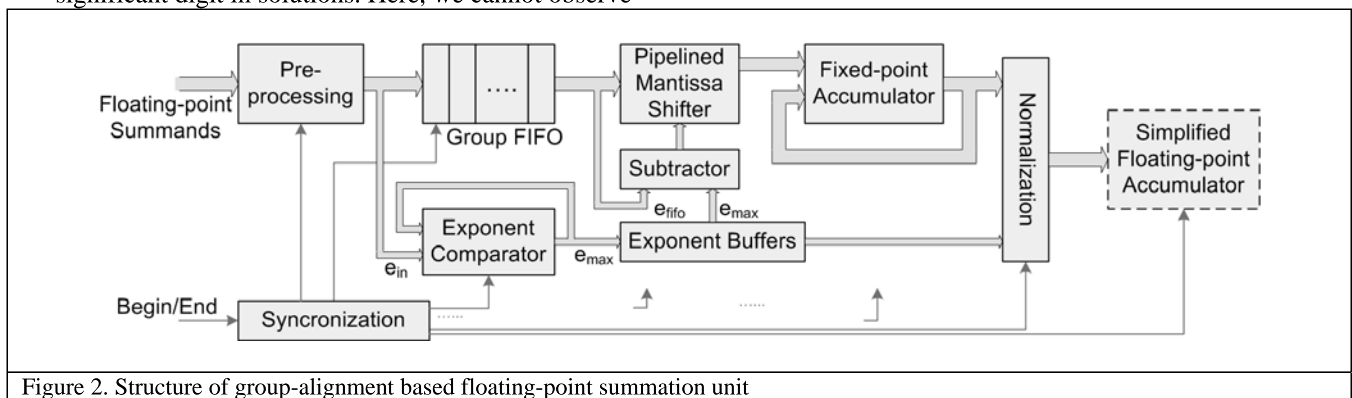

Figure 2. Structure of group-alignment based floating-point summation unit

Following features make our design distinct from others [13] [14]:

- To be compatible with conventional numerical computing software, the inputs and outputs of our summation unit are all in floating-point representations. But almost all internal stages use fixed-point arithmetic to save hardware resources as well as pipelining stages.
- Floating-point operands are feed into the single input port sequentially at a constant speed. Two local feed-back paths connect outputs of the single-cycle fixed-point exponent comparator and the mantissa accumulator with their inputs respectively so won't result in pipeline stalls.
- With the help of two external signals marking the beginning/ending of input groups or datasets, our design can always achieve the optimal sustained speed without the knowledge of summation length. Furthermore, multiple set of inputs can be accumulated consecutively with only one pipeline stall between them.
- The synchronization circuit automatically divide a long input dataset into small groups to take advantage of the group-alignment technique. Grouping is transparent to external and will not cause any internal pipeline stalls. The maximal size of a summand group is set to 16 in our implementation so that the corresponding group FIFO can be implemented efficiently by logic slices. The size of a group can also be reduced or enlarged to achieve better latency performance.
- Once the group FIFO contains a full summand group or the summation-ending signal is received, the synchronization circuit commands the exponent comparator to clear its content after sending the current value to the maximal exponent buffer. Starting from the next clock cycle, mantissas of those summands in FIFO are shifted out sequentially with the next group of operands moving-in. At the mean time, their exponents are subtracted from the corresponding maximal exponent one by one to produce differences for the pipelined mantissa shifter.
- The word-width of the barrel shifter is set to 34 bits (32 fraction bits) conservatively, and the fixed-point accumulator needs four more bits to prevent possible overflow. According to error analysis in Table 1, a 30-bit accumulator with 24 fraction bits is enough to achieve similar accuracy as the standard single-precision floating-point arithmetic.
- The single-cycle 38-bit group mantissa accumulator becomes the performance bottleneck of our design preventing us further improve its working speed. Instead of using wasteful Wallace-tree adder to remove the carry-chain from the accumulator's critical path [14], we simply disassemble the large fixed-point unit into two smaller ones. With their respective integer bits to prevent overflow, the resulting 21-bit fixed-point accumulator now can work at the high speed we required.
- The normalization circuit accepts outputs from the fixed-point accumulator(s) and exponent buffer, and then converts them into normalized floating-point format. It also consists of Leading-Zero-Detector (LZD) and pipelined left shifter as its correspondence in standard floating-point adders. However, because the data throughput of this stage is at least half of all front-end circuits, a more economic implementation can always be achieved.
- For long summations with multiple summand groups, another group summation stage using conventional pipelined floating-point adder is required to accumulate all group partial-sums. Because its data throughput is just 1/16 of the front-end, pipelining inside the adder will not cause any data-dependency problem. Furthermore, the foregoing normalization circuit and the floating-point adder can be combined together to save a costly barrel shifter as well as other unnecessary logics.
- This design can be easily extended to accept a whole group of summands through multiple input ports. In order to achieve the ideal data throughput as one group summation per clock cycle, multiple exponent comparators and fixed-point adders are parallelized in binary-tree structure, and The group FIFO is not necessary now.

Table 2. Comparison of single-precision accumulators

	Group-alignment ①	Scheduling [13]	Delayed-addition [14]
Target device	Virtex II Pro	Virtex II Pro	Virtex-E
Area (Slices)	443 (716)	633 ($n=2^4$) ~900 ($n=2^{16}$) ②	1095 CLBs
Speed (MHz)	250	180 ($n=2^4$) ~160 ($n=2^{16}$)	150
Pipeline stages	14 (23)	20	5 ③
Latency (cycles)	n<16: 2n+12 (21) n>16: 44 (53)	$\le 3\cdot(n+20\log_2 n)$	5 + 46ns
Summation length constraints	No	Yes ④	No

① Two numbers are listed at some places in this column for without and (with) the final group summation stage.
② One SRAM block is required for data buffering.
③ The final addition and normalization stage uses combinational logic, so is not pipelined.
④ n is set to a power of two.

Table 2 lists the performance of our single-precision floating-point summation unit together with two others proposed in [13] and [14]. They are compared with each other based on sustained FLOPS performance, internal

buffer requirements, latencies, etc. We observe that this new floating-point/fixed-point hybrid summation unit can provide us much higher computational performance, less RC resource occupations, as well as practical latency than previous designs. Furthermore, choosing more fraction bits for the fixed-point accumulator consumes negligible extra RC resources, but can significantly improve numerical error bounds of the summation.

6. Conclusion

In this paper, we propose a group-alignment based hardware algorithm to improve the evaluation of floating-point summations on FPGA-base RC platform. The corresponding summation unit can be efficiently constructed with fixed-point arithmetic but achieve similar or even better relative and absolute rounding errors than standard floating-point arithmetic. Comparing with previous designs, the new fully-pipelined summation unit can provide us much higher data throughput and less latency with much less hardware resources consumed.

Future work will concentrate on designing and analysing accurate arithmetic units based on the group alignment technique for bad-conditioned numerical computing applications. For example, this new summation unit can be easily applied to other linear algebra routines like vector norm, dot product, and matrix-vector multiplication improving their accuracy and stability significantly. Furthermore, we can also construct a tree-like parallel summation unit with the same technique if multiple summands can be supplied simultaneously. This approach can be applied to accelerate finite difference based numerical applications [19], and result in more efficient and compact implementations with much higher computational performance.

References

[1] B. Fagin and C. Renard, Field Programmable Gate Arrays and Floating Point Arithmetic, IEEE Transactions on VLSI Systems, 2(3), 1994

[2] K. D. Underwood, FPGAs vs. CPUs: Trends in peak floating-point performance, In proceedings of the ACM/SIGDA 12th international symposium on FPGA, 171-180, 2004

[3] P. Belanovic and M. Leeser, A Library of Parameterized Floating-point Modules and Their Use, In the Proceedings of the International Conference on Field-Programmable Logic and Applications, 2002

[4] J. Liang, R. Tessier, and O. Mencer, Floating-point unit generation and evaluation for FPGAs, In proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 185-194, 2003

[5] G. Govindu, L. Zhuo, S. Choi, and V. K. Prasanna, Analysis of high-performance floating-point arithmetic on FPGAs, In Proceesings of the 11th Reconfigurable Architectures Workshop, 2004

[6] A. A. Gaar, W. Luk, P. Y. Cheung, N. SHirazi, and J. Hwang, Automating customisation of floating-point designs, In the Proceedings of the International Conference on Field-Programmable Logic and Applications, 2002

[7] E. Roesler and B. Nelson, Novel Optimizations for Hardware Floating-point Units in a Modern FPGA Architecture, In the Proceedings of the International Conference on Field-Programmable Logic and Applications, 2002

[8] M. P. Leong, M. Y. Yeung, C. K. Yeung, C. W. Fu, P. A. Heng, and P. H. W. Leong, Automatic floating tofixed point translation and its application to post-rendering 3D wrapping, In proceedings of the Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 240-248, 1999

[9] W. Chen, P. Kosmas, M. Leeser, and C. Rappaport, An FPGA Implementation of the Two Dimensional Finite Difference Time Domain (FDTD) Algorithm, 12th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), 2004

[10] U. Kulisch, The Fifth Floating-Point Operation for Top-Performance Computers, Universitat Karlsruhe, 1997

[11] D. Goldberg, What every scientist should know about floating-point arithmetic, ACM Computing Surveys, 23(1):5-48, 1991

[12] D. Priest, Differences among IEEE 754 Implementations, http://www.validgh.com/goldberg/

[13] Ling Zhuo, Gerald R. Morris, Viktor K. Prasanna. "Designing Scalable FPGA-Based Reduction Circuits Using Pipelined Floating-Point Cores," 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), 2005

[14] Z. Luo and M. Martonosi, Accelerating pipelined integer and floating-point accumulations in configurable hardware with delayed addition techniques, IEEE Transactions on Computers, 49(3): 208-218, 2000

[15] N. J. Higham, The accuracy of floating point summation, SIAM Journal of Scientific Computing, 14: 783-799, 1993

[16] X. S. Li, J. W. Demmel, D. H. Bailey, G. Henry, et. al., Design, Implementation and Testing of Extended and Mixed Precision BLAS, ACM Transactions on Mathematical Software, 18(2): 152-205, 2002

[17] N. J. Higham, Accuracy and Stability of Numerical Algorithms, SIAM, Philadelphia, 2002

[18] www.xilinx.com/univ/xupv2p.html

[19] C. He, W. Zhao, M. Lu, Time Domain Numerical Simulation for Transient Waves on Reconfigurable Coprocessor Platform, In proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), 2005