A Floating-point Accumulator for FPGA-based High Performance Computing Applications

Song Sun Joseph Zambreno

Department of Electrical and Computer Engineering,
Iowa State University
Ames, Iowa, USA
{sunsong,zambreno}@iastate.edu

***Abstract*—A floating-point accumulator for FPGA-based high performance computing applications is proposed and evaluated. Compared to previous work, our accumulator uses a fixed size circuit, and can reduce an arbitrary number of input sets of varying sizes without requiring prior knowledge of the bounds of summands. In this paper, we describe how the adder accumulator operator can be heavily pipelined to achieve a high clock speed when mapped to FPGA technology, while still maintaining the original input ordering. Our experimental results show that our accumulator design is very competitive with previous efforts in terms of FPGA resource usage and clock frequency, making it an ideal building block for large-scale sparse matrix computations as implemented in FPGA-based high performance computing systems.**

I. Introduction

Floating-point accumulators (and vector reduction operators in general) form the basis of many scientific computing applications, and can be found in such diverse fields as weather forecasting, nuclear physics simulations, and image processing [1]. Given an input vector S, an accumulator performs a simple summation of the individual elements:

$$S = \sum s_i \tag{1}$$

While this is a trivial computation to describe mathematically, the need for fast floating-point accumulators (along with other operators) led in part to the emergence of vector processing in the 1980s. Accumulation is so commonplace in high performance computing applications that it has been described by some as the “fifth floating-point operation” [2].

One common computing scenario that requires a high performance accumulator is in Sparse Matrix-Vector Multiplication (SMVM) [3]. For example, Google’s PageRank eigenvector problem is arguably the world’s largest matrix computation (there were approximately 25 billion pages on the Internet in 2008) [4]. The PageRank algorithm is dominated by SMVM computations given that the Google matrix is very large (billions of rows and columns) and very sparse. Clearly, the performance of SMVM and by extension, floating-point accumulation has real-world impact.

Special care must be taken when designing FPGA-based accelerators for applications that make use of floating-point accumulators. The aforementioned SMVM operation calculates $Y = AX$, where $y_i = \sum_{j=1}^{n} a_{ij} x_j$ must be computed

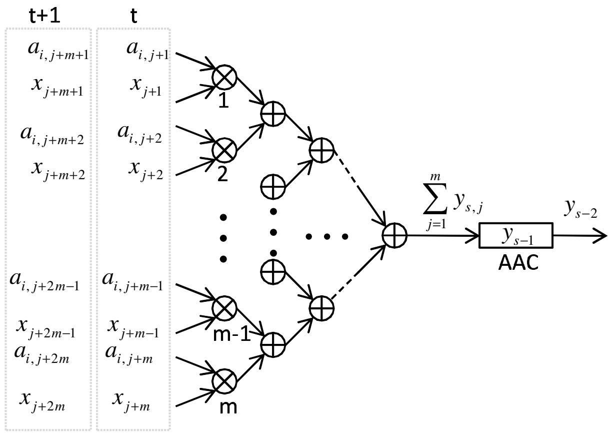


Fig. 1. Matrix-vector Multiplication Example

efficiently. A simple circuit to compute y_i is shown in Figure 1. In the case of a very large n (as is the case for Google PageRank), the elements in one row have to be partitioned into $\lceil \frac{n}{m} \rceil$ blocks. One block is sent to the circuit in each clock cycle. The partial sum $y_i^r = \sum_{j=(r-1)m+1}^{j=rm} y_{ij}$ produced in each clock cycle is accumulated near the output of the circuit where $y_i = \sum_{r=1}^{\lceil n/m \rceil} y_i^r$. The circuit module performing equation (1) is called the Adder Accumulator (AAC) or alternatively reduction circuit in [5], [6], [7]. The specific design we propose in this paper is referred to as a Floating-point Adder Accumulator (FAAC), to distinguish it from other AAC circuits that use fixed-point arithmetic.

The performance of any FPGA design is highly dependent on the clock frequency at which the circuit can run without timing constraint violations after placement and routing. The clock frequency in turn is determined by the delay of the critical path in the circuit. Adding two values in one clock cycle, as shown in Figure 2(a), will increase the critical path delay thus leading to a lower clock frequency. Such a single-cycle FAAC design would severely reduce the performance of the whole FPGA system. Consequently, any high performance FPGA-based accumulator must partition the floating-point addition across multiple clock cycles in order to improve the resulting clock frequency.

Also, as highlighted by [8], FPGA I/O capacity can often

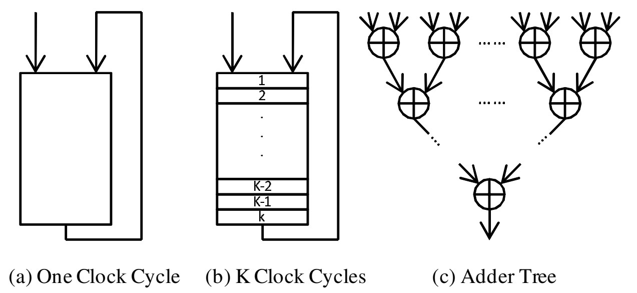


Fig. 2. Impractical FAAC Architectures

limit the performance of the system. Designing a scalable and pipelined stall-free accumulator to fully utilize the I/O bandwidth and computing resources is critical to achieve a high performance goal. Designing a floating-point accumulator with pipeline stalls is trivial. One straightforward way is to use a standard k-stage floating-point adder as shown in Figure 2(b). There are k clock cycles stalls between two additions. In other words, the sum of the first addition to be added with a third value will not come out of the adder until k clock cycles after the first two values entering the pipelines.

Removing these pipeline stalls is a challenging task since the pipeline input fed by the later pipeline output introduces feedback. That is, a new input data cannot be processed until the previous sum comes out from the end of the pipeline. An intuitive approach in designing a deeply pipelined and stall-free floating-point accumulator is the adder tree as shown in Figure 2(c). An adder tree with $n-1$ adders can be used to accumulate n floating-point values. The first row in the tree has $\lceil \frac{n}{2} \rceil$ adders; the i^{th} row has $\lceil \frac{n}{2^i} \rceil$ adders; the last row in the tree has only one adder whose output is the accumulated result. However, this approach is infeasible when n is large or the n input values cannot arrive in the same clock cycle. The resource utilization on a single FPGA device will be prohibitively high when n is large. Also, the number of pipeline stages to compute a sum will be intolerably large since the depth of the adder tree is $\lceil log_2^n \rceil \times k$, where k is the number of pipeline stages in an adder.

Our FAAC design as described in this paper accumulates positive and negative numbers separately so that the accumulation adder does only additions of numbers of the same sign. Saving leading zero count and cancellation shift reduces the latency of the adder. A subtractor which computes the difference of the two partial sub-sums is not in the accumulator loop. By using the log-sum technique, two standard adders reduce intermediate sub-sums. The summation order is different from the order of the inputs, the impact of this feature on result accuracy is discussed in Section VI. Preliminary results show that our heavily pipelined (48 stages) FAAC design can obtain high clock frequencies (and subsequently high performance) while using a relatively small amount of FPGA resources.

II. Problem Formulation

The input of the FAAC is a sequence of floating point numbers which may be positive or negative. The numbers belong to different groups. The numbers in the same group are fed into the FAAC consecutively. At each clock cycle, only one number enters the pipelines. More formally, suppose the input data stream is $\{a_{1,1}, a_{1,2}, ..., a_{1,n_1}\}$, $\{a_{2,1}, a_{2,2}, ..., a_{2,n_2}\}$, ..., $\{a_{i,1}, a_{i,2}, ..., a_{i,n_i}\}$,..., where $a_{i,j}$ is the j^{th} data in the i^{th} group. The numbers in the bracket belong to the same group. If $a_{s,j}$ arrives earlier than $a_{t,j}$, all the data in the s^{th} group must arrive earlier than the data in the t^{th} group. The problem is to sum up all the data in a group into a single value such that $a_i = \sum_{j=1}^{n_i} a_{i,j}, i = 1, 2, ...$ In the extreme case, each group has only one data value and the accumulator behaves as a queue.

III. Previous Work

The design of floating-point accumulators can be traced back to the 1980s with the emergence of pipelined vector machines [9], [1]. An adder accumulator adopting the architecture in [1] is well-suited for accumulating one input group. However, the buffer will overflow for multiple input groups. In [6], the reduction circuit uses only one adder with $\Theta(log(n))$ space and $\Theta(log(n))$ latency. However, the design can not handle the accumulation correctly when the number of inputs in a group is not a power of 2. The reduction circuit in [5] uses only one floating-point adder but needs two buffers of size k^2 where k is the number of pipeline stages of the adder. However, the latency of reducing one input group depends on the sizes of subsequent groups and the latency of reducing may be unacceptably large if the size of the subsequent groups is very large. Also, the output order of the sums may not be the same as the input order of the input groups. The design trade-offs and different approaches for accumulators are studied in [7]. The accumulators presented in both [10] and [11] require the user to specify bounds on the values to be accumulated beforehand.

IV. Design Principle

To accumulate n numbers, one straightforward observation is that the number of partial sums is at most equal to the number of stages of the adder. At most k adders will be used where k is the number of pipeline stages; one is to produce the partial sum, the other $k-1$ adders form an adder tree to sum up the $k-1$ partial sums. Designing an accumulator using this approach is prohibitive in a single FPGA device when k is large. Since the data is sent to the accumulator in a serial fashion, we use the log-sum technique [9] to reduce the partial sums. To reduce k partial sums produced by the first adder, the pipeline needs only another $\lceil log_2^k \rceil$ adders and is totally independent of the intrinsic number of stages in those adders. Therefore, the number of stages in the first adder is critical and determines the number of adders to be used and the delay of the whole accumulator. Figure 3 shows the two cases for $k = 4$ and $k = 8$. Notice that the number of adders is not determined by the value of p.

The relationship between clock frequency and pipeline stages can be formulated as $Frequency \propto (Pipeline\ Stages/Total\ Delay)$. To maintain a high clock

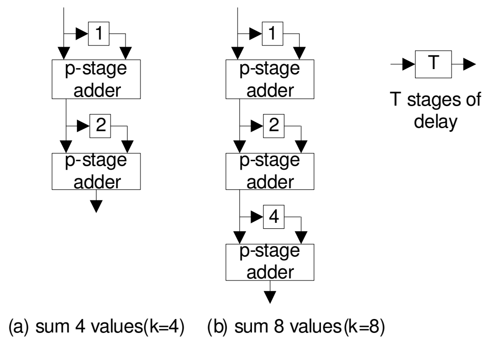


Fig. 3. Log-sum Technique

frequency while using fewer pipeline stages, we have to reduce the total delay of the adder. We observe that the floating point adder which only performs addition has a much lower delay than the adder which performs both addition and subtraction. For example, the former does not perform the mantissa comparison, the leading-zero counting and has fewer exception signalings and barrel shifters. Hence in this paper, we reduce the total delay by requiring that the first adder only performs addition.

The overview of our architecture is shown in Figure 4. The first adder performs addition on values of the same sign only. If the sign of the input is different from the sign of the output partial sum, the partial sum is pushed onto the stack with the same sign; otherwise, the partial sum acts as one of the inputs. The 4-stage adder accumulates any number of sequence values into at most 4 pairs of partial sums. The subtractor performing only subtraction produces at most 4 partial sums for an arbitrary number of values. Using the log-sum technique shown in Figure 3, another two standard adders are used to reduce the 4 intermediate partial sums. As shown in Figure 4, the first adder which only performs addition has $k = 4$ pipeline stages to produce the partial sums. In this paper we refer to these pipeline stages as $bands$. The increased number of pipeline stages will lead to a higher clock frequency. For example, an 8-band accumulator will typically have a higher frequency than a 4-band accumulator.

V. Architecture

Details of our proposed FAAC architecture are shown in Figure 5. It is mainly composed of three modules: the distributor, the subtractor and the full adder. The input data is sent to the distributor whose outputs are connected to the inputs of the subtractor. The outputs of the subtractor are in turn tied to the inputs of the full adder. The outputs of the accumulator are hard-wired to the outputs of the last full adder.

A. Distributor

The distributor is composed of a 4-stage adder, two stacks, and some control logic. One stack stores the positive data while the other stores the negative data. In each clock cycle, the incoming data is added with zero, output of the adder or the data on the top of the stack with the same sign. Since

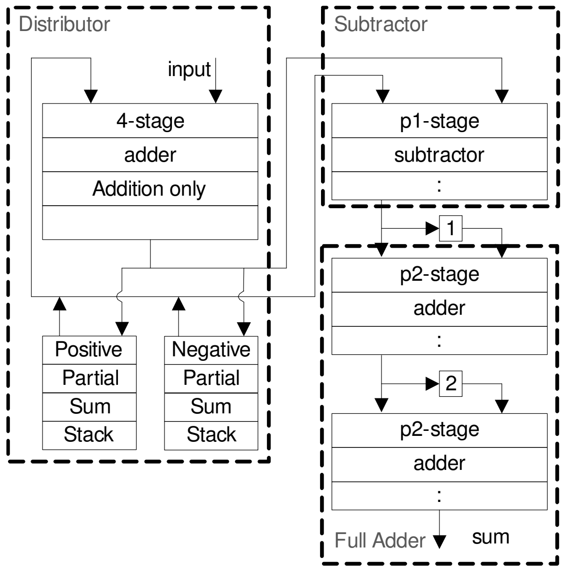


Fig. 4. Architecture Overview

the number of intermediate results for the adder is at most k, the depth of the stack is equal to k. Each input data has two auxiliary bits called the Input Control Codes (ICCs). The ICCF bit is equal to 1 if the input data is the first incoming number in a new group; otherwise, it is equal to 0. The ICCL bit is equal to 1 if the incoming data is the last number in a group; otherwise, it is equal to 0. If a group has only one data value, the two bits are both equal to 1. Two queues with the same depth as the 4-stage adder serve as FIFO buffers for the ICCs. Another queue $sign$ is for the sign bit of the input data since the signs of the two inputs are the same. The control logic is used to schedule the inputs of the 4-stage adder and the outputs of the distributor.

The input scheduling algorithm is shown in Figure 6. $input(in)$ is the incoming data; $input(reg)$ is the other input of the 4-stage adder. $last$ is the logic OR result of the ICCL queue elements. sum is the output of the 4-stage adder. $last = 0$ happens only when the current group has more than four numbers; $last = 1$ happens only when the last number of the current group is in the pipelines. In the latter case, the new incoming data should be added with zero and the output of the adder should not be added with the new incoming data, but be sent out as output as shown in Figure 7. This is because the incoming data and the output of the 4-stage adder belong to different groups. In the former case, the newly generated partial sum is added with the incoming data if they have the same sign; otherwise, it is pushed onto the stack and the new incoming data is added with a number popped off of the stack with the same sign. In the case that the stacks are empty or the incoming data is one of the first four numbers in a group, zero is added with the incoming data.

The distributor has two data outputs in each clock cycle; one is positive pos, the other is negative neg. One of the outputs must come from the partial sum, the other is popped by the stack with the opposite sign or zero when that stack

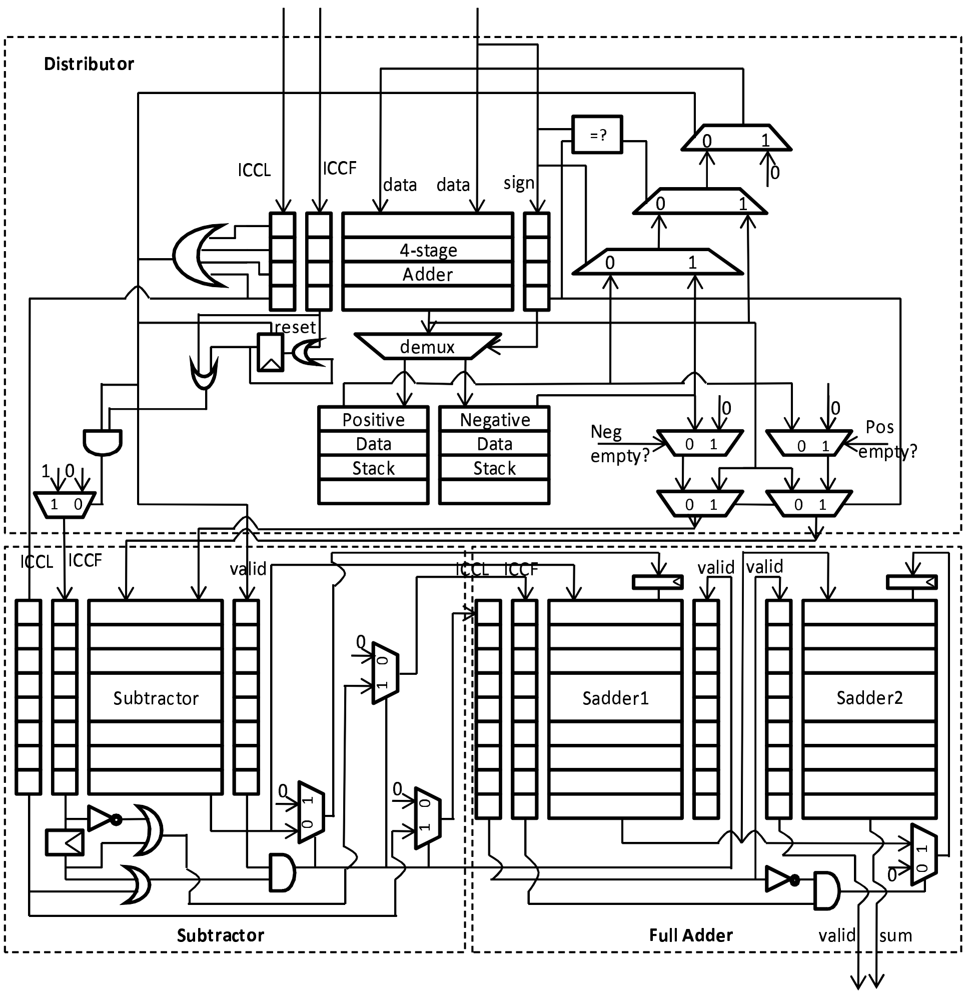


Fig. 5. Architectural Details of a 4-band FAAC

is empty. One of the outputs must be zero when the group being accumulated has less than five numbers. As shown in Figure 5, we use multiplexers to express the if statements in the algorithms.

The other three outputs of the distributor are valid, ICCF and ICCL. The valid signal is set when the data output should be treated as valid outputs. The ICCL/ICCF signal is set if the output is the last/first data pair of a group. In Figure 5, the valid signal is directly connected to $last$. The reason is that the output scheduling algorithm is triggered whenever $last$ is set. The ICCL signal is directly connected to the output of the ICCL queue since the last data of a group must be in the last output pairs.

The output ICCF signal should not be set until the valid outputs are available. Moreover, it should be set only when the current output pair is the first one of a group. There are two situations in the latter case. In the first situation, the number of data in the group is no more than four. The first partial sum is in the first output pair. Therefore, the output of the ICCF queue can be used as the output ICCF directly. That is, $ICCF_{out}^{t_i} = valid^{t_i} \bigcap ICCF_{queue}^{t_i} = last^{t_i} \bigcap ICCF_{queue}^{t_i}$, where $\bigcap$ and $\bigcup$ are logical AND and OR operations, respectively. In the second situation, the number of data in the group is larger than four. The first partial sum is not in the first output pair. However the circuit must remember that the first partial sum has been generated. A D flip-flop is used to serve as the memory cell. It is set once the first partial sum comes out of the pipeline and cleared once a new group comes out. That is, $ICCF_{out}^{t_i} = last^{t_i} \bigcap (\bigcup_{t=t_s}^{t=t_i} ICCF_{queue}^{t})$, where t_s is the clock cycle at which the first partial sum comes out of the adder pipeline; Obviously, $ICCF_{queue}^{t_s} = 1$. Combining the two cases together, we have $ICCF_{out}^{t_i} = last^{t_i} \bigcap (\bigcup_{t=t_s}^{t=t_{i-1}} ICCF_{queue}^{t} \bigcup ICCF_{queue}^{t_i})$.

B. Subtractor and Full Adder

The input of the subtractor in each clock cycle is a pair of positive and negative data values. It only performs floating-point subtraction. The full adder component has two floating-point full adders (also called Sadders) which can perform addition and subtraction. The input of the full adders can be positive or negative.

One input of Sadder1 is hard-wired to the output of sub-

```
**Algorithm**: Input Scheduling Algorithm
if $last = 0$ then
    if $sign(3) = sign(in)$ else
        $input(reg) := sum;$
    else
        $push\ sum\ into\ stack(sign(3));$
        if $stack(sign(in))\ is\ empty$ then
            $input(reg) := 0;$
        else
            $input(reg) := pop\ stack(sign(in));$
        end if;
    end if;
else
    $input(reg) := 0;$
end if;
```

Fig. 6. Input Scheduling Algorithm of Distributor

TABLE I
VALID INPUT PATTERNS OF SADDER1 ($k = 4$)

Left Input	First,Last	Last	Second	Last	Last
Right Input	0	First	First	0	Third
Input Number	1	2	3,4	3	4

tractor directly. The other input has one extra flip-flop used strictly as a single clock cycle delay. The number of outputs from the subtractor for each group is at most k. The Sadder1 will sum up at most two outputs at each clock cycle. The valid input patterns of the Sadder1 in the case of $k = 4$ are shown in Table I. The left and right inputs correspond to the inputs of the Sadder1 in Figure 5. In the first column of Table I, there is only one number which is both the last and first one of a group. The input pattern in the third column happens when the number of outputs from the subtractor is 3 or 4. Observing Table I carefully, the input pattern is valid when the last data is the left input or the first data is the right input and the second data is the left input. Hence the $valid$ input signal for the sadder in clock cycle t can be expressed as $valid^t_{in} = valid^t_{out} \bigcap (ICCL^t_{out} \bigcup (ICCF^{t-1}_{out} \bigcap \overline{ICCF^t_{out}}))$ where $valid_{in}$ is the input signal of Sadder1; $valid_{out}$ $ICCL_{out}$ $ICCF_{out}$ are the output signals of the subtractor. The right input must be cleared once a pair of valid inputs is sent to Sadder1. The corresponding circuit is shown in the bottom-left of Figure 5. For $k > 4$ where k is power of 2, we need to deal with two input numbers which contain neither the first nor the last data. For example, the input pattern which has the 5th and 6th inputs from the subtractor is valid while the input pattern with the 4th and 5th inputs is not valid. A single memory bit ov can be used to indicate whether the output of the subtractor has the even sequence order. It is also reset after a valid pair of data is sent to the Sadder; otherwise, it is flipped every clock cycle. The valid signal in this case can be expressed as $valid^t_{in} = valid^t_{out} \bigcap (ov^t \bigcup ICCL^t_{out} \bigcup (ICCF^{t-1}_{out} \bigcap \overline{ICCF^t_{out}}))$

```
**Algorithm**: Output Scheduling Algorithm
if $last = 1$ then
    if $sign(3) = +$ then
        $pos := sum;$
        $neg := pop\ stack(-);$
    else
        $neg := sum;$
        $pos := pop\ stack(+);$
    end if;
end if;
```

Fig. 7. Output Scheduling Algorithm of Distributor

The ICCL input of Sadder1 is connected to the ICCL output signal of the subtractor only when the input pattern is valid; otherwise, it is cleared. The ICCF input of Sadder1 is set if the ICCF output is set in the previous or current clock cycle and the input pattern is valid. For $k > 4$, the circuit for ICCL and ICCF inputs of Sadder1 is also correct.

The connection between Sadder1 and Sadder2 is similar to that of Sadder1 and the subtractor. In the $k = 4$ case, each group has at most two inputs for Sadder2. The input pattern is valid only if the ICCL input is set. The right input is cleared after a pair of valid inputs is sent to Sadder2.

VI. DATA ACCURACY

The accumulator performs the summation in a different order than the order of inputs. Since floating point addition is not truly associative, the accumulated result may not be compliant with the result of sequential floating point additions. In floating point addition, adding a large number with a tiny number will lose precision. For example, the sum of E80 and 2 is still E80 where 2 is shifted out of range to be 0. For a sequence of input (E80,-E80,2,-1), our design returns 0 while an in-order accumulator would return the exact result 1; for a sequence of input (E80,2,-E80,-2), an in-order accumulator returns -2 while our design returns the exact result 0. The value of the positive or negative sub-sum in our design will never decrease. Thus a sequence which would not overflow if summed sequentially may overflow in our design. This problem could be lessened by adding extra bits to the exponent and mantissa. Consider a long sequence of data whose sum converges to 0. The numbers arriving later become smaller and smaller. Since the positive and negative sub-sums in our accumulator will never decrease, the small number will be shifted out as 0. If the sum of the sequence converges to 0, our accumulated result will not be as close to 0 as the result of a sequential accumulator.

VII. AREA AND PERFORMANCE EVALUATION

A. General Analysis

In this section, the characteristics of the FAAC are compared with those designs proposed in [7]. Table II summarizes the different characteristics of our design and those in the previous

TABLE II
COMPARISON WITH PREVIOUS WORK

Design	No. of Adders	Buffer Size	Clk Frequency	Total Latency	Latency per Group	Out-of-order
PCBT [7]	$\lceil log_2^n \rceil$	$2\lceil log_2^n \rceil$	Decrease with n	$n + \alpha\lceil log_2^n \rceil$	Predictable	No
FCBT [7]	2	$3\lceil log_2^n \rceil$	Decrease with n	$\leqslant 3n + (\alpha - 1)\lceil log_2^n \rceil$	Predictable	No
DSA [7]	2	$\alpha\lceil log_2^n + 1 \rceil$	stable	$n + (\alpha - 1)\lceil log_2^\alpha + 1 \rceil$	Not Predictable	Yes
SSA [7]	1	$2\alpha^2$	stable	$\leqslant n + 2\alpha^2$	Not Predictable	Yes
FAAC	$< 2 + \lceil log_2^k \rceil$	$2k$	stable	$n + k + \alpha\lceil (1 + log_2^k) \rceil$	Predictable	No

work [7]. n_i is the number of data values in a group whereas $n = \sum_{i=1}^{p} n_i$ is the total amount of data in p groups. p is the maximum number of groups which can be put into α segments [7]. In practical applications (e.g. SMVM), n is usually much larger than n_i which in turn is much larger than k. As mentioned in [7], the practical value of n is over tens of thousands and α is under 20 while k in our design is only 4.

As discussed in Section IV, the number of adders to accumulate k partial sums is $\lceil log_2^k \rceil$. Suppose the adder which only performs addition in our architecture has k stages instead of 4. The number of adders we use are the k-stage adder, the subtractor and $\lceil log_2^k \rceil$ full adders which is less than $2 + \lceil log_2^k \rceil$ adders. The buffers used in the FAAC are two stacks whose sizes are equal to $2k$.

The delay to accumulate a set of groups is referred to as the *total latency* which starts from the clock cycle at which the first number of the first group comes into the accumulator until the clock cycle at which the sums for all groups come out of the pipeline. The *latency per group* is the latency to accumulate a group with n_i numbers starting from the clock cycle at which the first number enters into the accumulator until the clock cycle at which the sum comes out of it. Suppose the last data in a group entered into the accumulator at clock cycle t_s. The sum of the numbers in the group is ready in clock cycle $t_s + C$ where C is the sum of the pipelines stages in the FAAC. In our implementation, $C = 4 + 14 + 15 + 15 = 48$. If a full adder has α pipeline stages and the subtractor is considered to be a full adder, C is equal to $k + \alpha + \alpha\lceil log_2^k \rceil = k + \alpha\lceil (1 + log_2^k) \rceil$. The *latency per group* of the FAAC is therefore only dependent on the amount of data in the current group and equal to $n_i + C$. In each clock cycle, input data enters into the FAAC without stalls. It takes n clock cycles for the last number in the last group to enter into the FAAC. After C clock cycles, the sum of the last group is ready. Therefore, the *total latency* of the FAAC is $n + C$.

As mentioned in [7], the PCBT design is infeasible to be implemented in an FPGA due to the large number of floating-point adders. From Table II, the area of PCBT, FCBT, and DSA increases with the number of data values to be accumulated; SSA has the smallest number of adders while the FAAC has the smallest buffer size. With the rapidly increasing gate count in modern FPGAs, the number of adders in the FAAC which grows logarithmically with k is acceptable. As we will see later in this section, the FAAC circuit with $k = 4$ occupies only a small fraction of a Xilinx Virtex 5 device

TABLE III
CHARACTERISTICS OF FAAC MODULES ON XILINX FPGAS

Module	Pipe Stages	XC2VP30		XC5VLX110T	
		Slices (%)	MHz	Slices (%)	MHz
Adder	4	548 (4%)	162	267 (1.5%)	244
Distributor	4	1303 (9%)	162	562 (3%)	244
Subtractor	14	1367 (9%)	187	494 (2%)	297
Sadder1	15	1806 (13%)	176	740 (4%)	286
Sadder2	15	1798 (13%)	191	702 (4%)	280
Total:	48	6252 (45%)	162	2269 (13%)	244

which makes it suitable to be used as a building block in a larger FPGA-based system on the same chip.

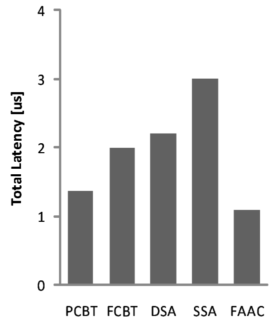
The clock frequency of PCBT and FCBT will decrease dramatically with n [7]. Thus, they are not proper choices to accumulate large sets of data. The PCBT and FCBT designs cannot start the computation without knowing the maximum size of datasets beforehand. The *total latency* of DSA and SSA depends on not only the amount of data in the current group but also on those of previous and subsequent groups. The unpredictable latency of DSA and SSA imposes great difficulty when applying them to large-scale SMVM and other applications. The dependency comes from the fact that DSA and SSA schedule inputs from different groups to the adders. The previously arrived group whose size is very large might still be coalescing while a small group finishes. This also results in another drawback of the DSA and SSA designs that the order of the sums is different from the order of the input groups. The authors in [7] suggest writing the outputs into a buffer and reading them out in order. An additional tag has to be attached to each group to differentiate them in the end. In contrast, the FAAC has a reasonable and predictable latency for each group. The latency for a group in the FAAC is only determined by the size of the current group and the total number of pipeline stages in the accumulator. The output order of the sums in the FAAC is the same as the input order of the groups which eliminates any post processing.

B. Design Implementations

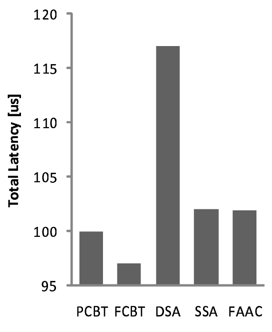
We made use of a double precision floating-point core from the OpenCores library [12], which we enhanced to perform accumulation using our own VHDL modules. Mentor Graphics Modelsim 6.4c and Xilinx ISE 10.1 were used for simulation and implementation. The proposed architecture was implemented in both a Xilinx Virtex-V XC5VLX110T and a Xilinx Virtex-II Pro XC2VP30 FPGA. The characteristics of all components are listed in Table III. The *Adder* is the 4-

TABLE IV
IMPLEMENTATION CHARACTERISTICS (n=128, $\alpha = 14$, $k = 4$)

Design	Adders	Slices	BRAMs	Clk Freq	Latency
PCBT	7	6808	–	165 MHz	226
FCBT	2	2859	10	170 MHz	$\leqslant 475$
DSA	2	2215	3	142 MHz	232
SSA	1	1804	6	165 MHz	$\leqslant 520$
FAAC	< 4	6252	0	162 MHz	176



(a) a single set of 128 inputs



(b) 128 sets of 128 inputs

Fig. 8. Latency for Accumulating Datasets

stage adder which only performs addition. The $Subtractor$ is the adder which only performs subtraction. The stages of the $FAAC$ are the sum of the stages in $Distributor$, $Subtractor$ and two $Sadders$. However, the area of the $FAAC$ is larger than the sum of the components since the multiplexers and logic gates shown in Figure 5 are not included in those components. In the Virtex-II Pro implementation, no more than 1% of the area is used for control logic. The total area of the FAAC is less than 4 full adders. In this initial implementation, only the $rounding\ toward\ zero$ mode is supported. Even though the 4-stage adder has a low delay, the critical path of the FAAC lies in the 4-stage adder of the distributor.

We compared our architecture implemented in the Virtex-II Pro device with those designs in [7]. The characteristics of the different designs are summarized in Table IV where $n = 128$, $\alpha = 14$, and $k = 4$. The latency is computed according to clock cycles based on the formula provided in Table II. Though the FAAC uses no more than four adders, the number of the slices it uses is almost the same as that of PCBT. This is due in part to the fact that we used an inefficient open-source floating-point core. Shrinking the resource usage of the adder is part of our future work.

C. Performance Comparison

Figure 8(a) shows the total latency of the designs for accumulating a single set of 128 inputs. Even though the clock frequency of the FAAC is not the highest, it achieves the smallest latency to reduce a single set. This is because the FAAC requires the smallest number of clock cycles. Figure 8(b) shows the total latency for accumulating 128 sequential sets of 128 inputs. The FAAC has larger latency than PCBT and FCBT since the clock period dominates the latency with a large number of inputs. However as mentioned in [7], PCBT is not realistic for most FPGA-based designs; PCBT and FCBT must know the largest size of datasets beforehand.

VIII. CONCLUSION

In this paper we described an area and performance efficient architecture for a floating-point adder accumulator which can be used to reduce an arbitrary number of groups with arbitrary sizes. Compared with previous work, our approach has several advantages: (1) the area and the clock frequency remain unaffected by the inputs; (2) the accumulation time for each group does not depend on any other group; (3) the number of clock cycles to accumulate a group is equal to the sum of the group size and the number of pipeline stages in the FAAC; (4) the output order of the sums is the same as the input order of the groups; (5) the design does not require any preprocessing of the input data. Besides those advantages mentioned above, the performance analysis shows that our architecture is very competitive compared with other designs.

Though the area usage of the FAAC is greater than most of profiled existing designs, the total amount of area fits comfortably into current-generation (Virtex 5) and older-generation (Virtex-II Pro) mid-range FPGA devices. We believe our FAAC circuit is a good choice for FPGA-based high performance computing applications.

REFERENCES

[1] L. M.Ni and K. Hwang, "Vector-reduction techniques for arithmetic pipelines," *IEEE Transactions on Computers*, vol. c-34, no. 5, pp. 404–411, 1985.
[2] U. Kulisch, "The fifth floating-point operation for top-performance computers," Universitat Karlsruhe, Tech. Rep., 1997.
[3] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel, "Optimization of sparse matrx-vector multiplication on emerging multicore platforms," in *Proceedings of Supercomputing (SC)*, 2007.
[4] S. McGettrick, D. Geraghty, and C. McElroy, "An FPGA architecture for the PageRank eigenvector problem," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, 2008, pp. 523–526.
[5] L. Zhuo and V. Prasanna, "High-performance and area-efficient reduction circuits on FPGAs," in *Proceedings of the International Symposium on Computer Architecture on High Performance Computing (SBAC-PAD)*, 2005, pp. 52–59.
[6] L. Zhuo, G. Morris, and V. Prasanna, "Designing scalable FPGA-based reduction circuits using pipelined floating-point cores," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, 2005.
[7] ——, "High-performance reduction circuits using deeply pipelined operators on FPGAs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 10, pp. 1377–1392, 2007.
[8] K. D. Underwood, K. S. Hemmert, and C. Ulmer, "Architectures and APIs: assessing requirements for delivering fpga performance to applications," in *Proceedings of Supercomputing (SC)*, 2006.
[9] P. M. Kogge, *The Architecture of Pipelined Computers*. Hemisphere Publishing Corporation, 1981.
[10] F. de Dinechin, B. Pasca, O. Cret, and R. Tudoran, "An FPGA-specific approach to floating-point accumulation and sum-of-products," in *International Conference on Field-Programmable Technology (FPT)*, 2008, pp. 33–40.
[11] C. He, G. Qin, M. Lu, and W. Zhao, "Group-alignment based accurate floating-point summation on FPGAs," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, 2006, pp. 136–142.
[12] *Double Precision Floating Point Core*, www.opencores.org, OpenCores, 2009.