High-Performance and Area-Efficient Reduction Circuits on FPGAs *

Ling Zhuo and Viktor K. Prasanna
Department of Electrical Engineering
University of Southern California
Los Angeles, California, 90089-2560 USA
{lzhuo, prasanna}@usc.edu

Abstract

Field-Programmable Gate Arrays (FPGAs) have become an attractive option for scientific applications. However, due to the pipelining in the FPGA-based floating-point units, data hazards may occur during reduction of series of values. A typical example of reduction is the accumulation of sets of floating-point values, which is needed in many scientific operations such as dot product and matrix-vector multiplication. Reduction circuits can significantly impact the overall performance, impose unrealistic buffer requirements, or occupy large area on the FPGA. In this paper, we introduce a high-performance and area-efficient FPGA-based reduction circuit. It can reduce multiple sets of sequentially delivered floating-point values without stalling the pipeline. In contrast with previous works, the proposed circuit uses one floating-point adder, and can handle input sets of arbitrary size. The buffer size needed by the circuit is independent of the size of the individual sets and the number of input sets. Using a Xilinx Virtex-II Pro FPGA as the target device, we implement the proposed reduction circuit and present performance and area results.

1 Introduction

FPGAs are a form of reconfigurable hardware. They offer flexibility in design like software, but with time performance that can be closer to Application Specific Integrated Circuits (ASICs). Due to their low computing density, earlier FPGAs were mainly used for applications that were not computationally demanding. However, with the rapid advances in technology, current FPGA devices contain much more resources than their predecessors. FPGAs also have been employed to implement floating-point scientific applications, and have achieved performance highly competitive with general-purpose processors [9, 11, 16].

Dramatic increases in the computing power of FPGAs have aroused strong interests in the supercomputing industry. Several vendors have developed or are developing high performance reconfigurable computing systems. Such systems include SRC MAPstation [10] and Cray XD1 [2], among others. In these systems, general-purpose computing systems are combined with FPGAs which serve as hardware application accelerators.

However, the MHz-scale clock rate of FPGAs is relatively low compared with the GHz-scale clock rate of general-purpose processors. If high-performance is the motivation for using FPGAs, then the design must exploit algorithmic parallelism and use deeply pipelined floating-point units. Unfortunately, using pipelined floating-point units for data reduction, such as accumulation of sequentially delivered floating-point values, may cause data hazards.

The reduction problem arise in FPGA-based designs for many scientific applications. For example, in both dot product and matrix-vector multiplication, series of floating-point values need to be accumulated. As the source matrix or vectors are usually too large to be stored on the FPGA, they need to be read in sequentially from the external memory. Another example is the Lennard-Jones force calculation in molecular dynamics [9]. At the end of the application, values that are generated serially need to be accumulated. Thus, we need a high-performance and area-efficient reduction circuit that can reduce series of sequentially delivered floating-point values.

Some simple but inefficient solutions exist for the reduction problem. For example, we could use $n - 1$ adders for accumulating n numbers. However, it is prohibitive to implement $n - 1$ adders on a single FPGA device, due to their resource utilization and routing complexity. We could also buffer incoming values when the reduction is in process. In this case, the size of the buffer increases with the number of inputs. Several optimized reduction circuits have also been proposed. Some of them utilize special-

*This work is supported by the United States National Science Foundation under award No. CCR-0311823 and in part by award No. ACI-0305763.

ized adder-level techniques like delayed addition [5] and may lead to pipeline stalls. These stalls adversely affect the performance of the circuit, and may even negate the performance gains derived from hardware acceleration using FPGAs. Some of the circuits do not work efficiently with multiple input sets [8] or cannot handle arbitrary number of inputs [14]. Other designs can handle input sets of arbitrary size, but employ more than one floating-point adder [4, 7].

In this work, we propose a reduction circuit that can handle *multiple input sets of arbitrary sizes*, using one floating-point adder *only*. The circuit needs two buffers of size α^2, where α is the pipeline delay of the adder. For n inputs, the upper bound on the latency of the circuit is $n + 2\alpha^2$. We have implemented the circuit on a Xilinx Virtex-II XC2VP7 FPGA [13]. Note that the adder in the circuit can also be replaced by other binary operators for other types of reduction.

The rest of the paper is organized as follows. Section 2 presents a formal definition for the reduction problem and discusses prior work. Section 3 proposes our high-performance reduction circuit and illustrates its operations using examples. Section 4 presents the performance of our reduction circuit and discusses some applications of the circuit. Section 5 concludes the paper.

2 Background & Related Work

2.1 Problem Definition

We can define the reduction problem as follows: We are given p sets of inputs, with set i containing n_i floating-point values ($0 \leq i \leq p-1$). The values in the sets are delivered sequentially as $[(s_{0,0}, \cdots, s_{0,n_1-1}), \cdots, (s_{p-1,0}, \cdots, s_{p-1,n_p-1})]$. The problem is to reduce n_i values in set i into a single value such that $r_i = \sum_{j=0}^{n_i-1} s_{i,j}$, $i = 0, 1, \ldots p-1$.

Our goal is to solve this problem using one floating-point adder and buffers of fixed size. The solution should avoid any data hazard or pipeline stall. Throughout the paper, α denotes the number of pipeline stages in the adder.

2.2 Previous Work

Research on reduction circuits for pipelined architectures started a couple of decades ago, when pipelined computers and vector computers first became available. Kogge proposes a method which uses $\lg(n)$ adders to reduce multiple input sets, where n is the maximum size of the input sets [4]. In [8], the authors propose a reduction method which uses one adder and a buffer of fixed size. This method is well suited for reducing one input set. However, for multiple input sets, the method assumes these sets are all stored in the external memory. If the input sets are generated sequentially, the buffer will overflow.

We have proposed several reduction circuits suitable for implementation on FPGAs. The intuitive idea of our prior work is that the accumulation of a set can be abstracted as traversing a binary addition tree in level order. When the size of each input set is a power of 2, we have proved that we can perform the additions in all the levels of the tree using one floating-point adder [14]. This reduction circuit uses $\lg(n)$ buffers of fixed size, where n is the maximum size of an input set. The latency is $\Theta(n)$.

However, when the addition tree is not a complete binary tree, the utilization of the adder in the circuit in [14] may exceed 100%. We thus extended the design by employing a second floating-point adder [7]. One adder in the circuit performs additions in the lowest level of the tree, while the other adder performs additions in all the other tree levels. This circuit needs $\lceil \lg(n) \rceil$ buffers of fixed size, and completes in $\Theta(n)$ clock cycles. Another design in [7] also employs two floating-point adders, but is based on a different idea. It allows one adder to reduce all the values for a given set. After the last value of a set arrives, there are multiple partial reductions in the adder pipeline; another adder begins reducing the new set while the first adder completes reducing the previous set. The design uses one buffer of size $\alpha(\lceil \lg \alpha \rceil + 1)$.

In this paper, we propose a reduction circuit which is very different from the previous works. It uses *one floating-point adder only* and can reduce multiple input sets of *arbitrary* size.

3 Reduction Circuit

3.1 Intuitive Idea

For $n > \alpha$, we can reduce n inputs to α items using one adder, without causing read-after-write data hazards. To achieve this, we write the first α inputs into a buffer; in each of the subsequent clock cycles, one item in the buffer and the new input to the circuit are fed to the adder as operands; the output of the adder is written back to the buffer. After $n + \alpha$ clock cycles, all the inputs are reduced to α items.

However, due to data dependencies, reducing the resulting α items requires $\Theta(\alpha \lg(\alpha))$ clock cycles using a tree traversal. During this period, a buffer is needed to store the incoming inputs, and the buffer size increases with the number of input sets.

To reduce the buffer size, we need to utilize the adder more efficiently. Suppose α distinct sets are stored in a buffer and each set has α items. Then we can interleave the additions from the α sets so that the adder is fully utilized and no data hazard occurs. In this way, reducing α^2 items from α distinct sets at most takes α^2 clock cycles. At

the same time, another buffer of size α^2 is needed to store the new inputs. Thus, we can reduce multiple inputs sets with one adder and two buffers of size α^2.

3.2 Architecture

Based on the idea discussed above, we propose a new reduction circuit. The architecture is shown in Figure 1. It contains one floating-point adder and two buffers of size α^2. The input can either enter one of the buffers, or enter the adder directly. The adder selects its operands from the input and the buffers. If the output of the adder is the final result of a set, it is written to external memory; otherwise, it is written back to one of the buffers.

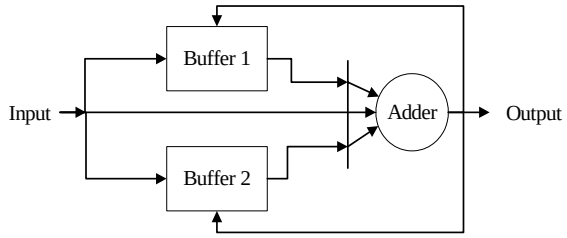


Figure 1. Architecture of reduction circuit

The buffer which accepts new inputs is denoted as Buf_{in}. For set i, if $n_i \leq \alpha$, n_i inputs are written into Buf_{in} directly; otherwise, α inputs are written into Buf_{in} and are then added with the remaining new inputs of the set. In the n_i (if $n_i > \alpha$) or α (if $n_i \leq \alpha$) cycles in which Buf_{in} accepts new inputs, the adder is used by another buffer. This buffer is called Buf_{red}, which stores no more than α items for each set. Reading from Buf_{red} column by column, the adder interleaves α additions from α distinct sets. The results of these additions are written back to Buf_{red}. Buffer 1 and Buffer 2 function as Buf_{in} and Buf_{red} alternately. When Buf_{in} is full, the two buffers are swapped. That is, Buf_{in} becomes Buf_{red} and Buf_{red} becomes Buf_{in}.

3.3 Algorithm

A. Basic Case We first consider a basic case, in which all sets have more than α inputs, that is, $n_i > \alpha$, $i = 0, 1, \ldots, p-1$. In this case, when Buf_{in} is full, it contains α distinct sets and each set contains α items. When this buffer becomes Buf_{red}, the adder reads intermediate results from it column by column. In this way, the adder interleaves additions from α distinct sets, and thus avoids any read-after-write hazards.

Figure 2 describes the operations of each component in the architecture during a clock cycle. It is assumed that the buffers allow read-after-write access. That is, if an entry is read and written in the same clock cycle, the output of the buffer is the value which is written.

1: **if** the input is among the first α inputs of a set **then**
2: write the input into Buf_{in};
3: if it is the first input of the set, then increment i
4: **if** Buf_{red} is not empty **then**
5: **if** $l = 0$ **then**
6: add $Buf_{red}[j\alpha + l]$ with zero
7: **else**
8: add $Buf_{red}[j\alpha + l]$ with $Buf_{red}[j\alpha + l - 1]$
9: **end if**
10: increment j
11: if $j + 1 = \alpha$, then $j = 0$, increment l
12: **end if**
13: **else**
14: add the input with $Buf_{in}[i\alpha + k]$, increment k
15: **end if**
16: {output of the adder}
17: **if** one operand is from $Buf_{in}[x]$ **then**
18: **if** the set is not the last set in Buf_{in} **then**
19: write the adder output to $Buf_{in}[x]$
20: **else**
21: write the adder output to $Buf_{red}[x]$
22: **end if**
23: **else if** operands are from $Buf_{red}[y]$ and $Buf_{red}[y-1]$ **then**
24: **if** these operands are the last two items of a set **then**
25: write the output to the external memory
26: **else**
27: write the output to $Buf_{red}[y]$
28: **end if**
29: **end if**
30: {buffers}
31: **if** the input is the last input of the last set in Buf_{in} or (no new input and Buf_{red} is empty) **then**
32: swap Buf_{in} and Buf_{red}
33: $j = 0, l = 0, k = 0, i = -1$
34: **end if**

Figure 2. Algorithm for the basic case

We now prove the correctness of the reduction circuit for the basic case. We first prove that the usage of the adder is collision-free, that is, only one buffer provides operands to the adder in a clock cycle. We also prove that the usage of the adder does not cause data hazards, and a buffer never needs to store more than α^2 items.

Theorem 1 *The algorithm shown in Figure 2 guarantees a collision-free and hazard-free use of the adder, and does not cause buffer overflow.*

Proof. During the execution of the algorithm, the adder only reads from Buf_{red} when Buf_{in} is accepting new inputs.

In such clock cycles, the adder does not read from Buf_{in}. Thus, the use of the adder is collision-free.

When Buf_{in} is full, each row of this buffer contains items of a single set. Otherwise, one of the sets must have (1) less than α items in Buf_{in}, which contradicts our assumption that $n_i > \alpha$ $(i = 0, \ldots, p-1)$; (2) more than α items in Buf_{in}, which is not allowed by the algorithm. Thus, when we interleave α additions from α rows, we interleave α additions from α distinct sets. Hence the use of the adder is hazard-free.

We next prove the buffers do not overflow. Initially, both buffers are empty. Without loss of generality, suppose Buffer 1 is used as Buf_{in}, and Buffer 2 is used as Buf_{red}. After the inputs of α sets enter the architecture, Buffer 1 is full and is designated as Buf_{red}. The algorithm then takes α^2 clock cycles to read from Buffer 1. During these clock cycles, the first α inputs of α consecutive sets are written into Buffer 2. When the remaining inputs of the sets enter the architecture, the adder accepts operands from Buffer 2, and not Buffer 1. After α sets enter the architecture, Buffer 1 is empty and Buffer 2 is full; then Buffer 1 is designated as Buf_{in}, and Buffer 2 is designated as Buf_{red}. The algorithm continues in this way until the last input enters the architecture. Thus, the buffers never overflow. ■

Theorem 2 *The reduction circuit correctly reduces multiple input sets. That is,* $r_i = \sum_{j=0}^{n_i-1} s_{i,j}, i = 0, \ldots, p-1$.

Proof. According to the algorithm, it is clear that each input or intermediate result goes through the adder once and only once. Thus, if we can prove that r_i only contains items of set i, we can prove the theorem.

Suppose during some clock cycle, the adder accepts two operands from different sets. In the algorithm, the adder always reads operands from two consecutive locations in the same row of Buf_{red}. If these items are from different sets, there must be a row which contains items from two different sets. However, we have shown that each row of Buf_{red} only contains items of one set. ■

The latency of reducing one input set depends on the sizes of subsequent sets. However, we can calculate the total latency for reducing p sets.

Theorem 3 *The reduction circuit reduces p sets in at most $(\sum_{i=0}^{p-1} n_i + 2\alpha^2)$ cycles.*

Proof. Since the reduction circuit never stalls while reading the inputs, the last input of the last set enters the reduction circuit in clock cycle $\sum_{i=0}^{p-1} n_i$.

At this time, if p is a multiple of α, Buf_{in} becomes full, and Buf_{red} is empty. Then the full buffer becomes Buf_{red}. From the next clock cycle onwards, the adder reads from Buf_{red}, and the last item of it is read in clock cycle $\sum_{i=0}^{p-1} n_i + \alpha^2$. Thus, the final result of the pth set is available in clock cycle $\sum_{i=0}^{p-1} n_i + \alpha^2 + \alpha$.

On the other hand, if $p \mod \alpha = p'$, $\alpha(\alpha - p')$ clock cycles are needed to empty Buf_{red}; then Buf_{in} becomes Buf_{red}. The operands for the last addition enter the adder in clock cycle $\sum_{i=0}^{p-1} n_i + \alpha(\alpha - p') + \alpha(\alpha - 1) + p'$. Thus, the final result of the pth set is available in clock cycle $\sum_{i=0}^{p-1} n_i + 2\alpha^2 - p'\alpha + p'$.

In both cases, the total latency is less than $\sum_{i=0}^{p-1} n_i + 2\alpha^2$ cycles. ■

B. General Case We next consider the general case, in which n_i can be of any value larger than 1. In this case, when Buf_{in} is full, it may contain more than α distinct sets. The problem now is how to interleave the additions in these sets so that data hazards do not occur.

We try to fit all the sets into α rows. This problem can be reduced to the *subset-sum problem* which is NP-complete [1], if an input set is not allowed to be partitioned among two rows, Therefore, we allow an input set to be split among two rows. Suppose the n_i $(n_i < \alpha)$ or α $(n_i \geq \alpha)$ items of set i are written into Buf_{in} in row-major order. We know that the items in a set can only exist in two consecutive rows. Otherwise, a set will have more than α items in Buf_{in}, which is not allowed by the algorithm.

The algorithm for the general case is shown in Figure 3. It is very similar to that for the basic case, with the following differences:

1. The first column of Buf_{red} is read twice. During the second read operation, $Buf_{red}[j\alpha]$ contains the intermediate result of the set which is split between row $j-1$ and row j $(j = 1, \ldots, \alpha - 1)$.

2. It is possible that an input set is split between two buffers. This case can be seen as the set is split among row 0 and row $0-1$. A special register is used to store the intermediate result from row $0-1$. It is then added to $Buf_{red}[0]$.

We now prove the correctness of the reduction circuit for the general case, and calculate the latency.

Theorem 4 *The algorithm shown in Figure 3 guarantees a collision-free and hazard-free use of the adder, and does not cause buffer overflow.*

Proof. The use of the adder is collision-free because the adder only reads from Buf_{red} when the Buf_{in} is accepting new inputs.

We next prove data hazards do not occur in the algorithm. When Buf_{in} is full, each column of the contains items from α distinct sets. Otherwise, there must be a set that contains more than α items in Buf_{in}, which is not allowed by the algorithm. Therefore, through interleaving additions in α distinct rows, we interleave additions in α distinct sets. Thus the use of the adder is hazard-free.

```
1: if the input is among the first α inputs of a set then
2:    write the input into Buf_in
3:    if it is the first input of the set, increment i
4:    if Buf_red is not empty then
5:       if l < α and (Buf_red[jα+l] and Buf_red[jα+l−1]
         are in the same set) then
6:          add Buf_red[jα + l] with Buf_red[jα + l − 1]
7:       else if l = α then
8:          if a set is split between row j−1 and row j then
9:             add Buf_red[(j − 1)α + l − 1] with Buf_red[jα]
10:         else if a set is split between two buffers then
11:            add Buf_red[0] with temp
12:         end if
13:      end if
14:      increment j
15:      if j + 1 = α, then j = 0, increment l
16:   end if
17: else
18:   adds the input with the kth item of set i in Buf_in,
      increment k
19: end if
20: {output of the adder}
21: if one operand is from Buf_in[x] then
22:   if it is not from the last set in Buf_in then
23:      write the adder output to Buf_in[x]
24:   else
25:      write the adder output to Buf_red[x]
26:   end if
27: else if one operand is the last item of a set in Buf_red
    then
28:   write the output to the external memory
29: else if operands are from Buf_red[y] and Buf_red[y − 1]
    then
30:   write the output to Buf_red[y]
31: else if one operand is from row j and is the last item of
    a set split between row j − 1 and row j then
32:   write the output to Buf_red[jα]
33: else if one operand is from row 0 and is the last item of
    a set split between two buffers then
34:   write the output to Buf_red[0]
35: else if one operand is from row α − 1 and is the last
    item of a set split between two buffers then
36:   write the output to temp
37: end if
38: {buffers}
39: if Buf_in is full or (no new input and Buf_red is empty)
    then
40:   swap Buf_in and Buf_red
41:   j = 0, l = 1, k = 0, i = −1
42: end if
```

Figure 3. Algorithm for the general case

In the algorithm, α^2 clock cycles are needed to fill Buf_{in}. Also, α^2 clock cycles are needed to empty Buf_{red}. Thus, we can prove the buffers do not overflow as in the proof of Theorem 1. ■

Theorem 5 *The circuit correctly reduces multiple input sets. That is,* $r_i = \sum_{j=0}^{n_i-1} s_{i,j}, i = 0, \ldots, p-1$.

Proof. We first prove that each set in Buf_{red} is reduced correctly. Clearly, this is true if set i is not split between two rows.

Suppose set i is split between row $j-1$ and row j, $j = 1, \ldots \alpha - 1$. The items in row $j-1$ are summed up and the intermediate result is written to $Buf_{red}[(j-1)\alpha+\alpha-1]$. If only one item of the set is stored in row j, it must be $Buf_{red}[j\alpha]$ because the items of a set are stored consecutively in row-major order. As $Buf_{red}[(j-1)\alpha+\alpha-1]$ is added with $Buf_{red}[j\alpha]$, the set is reduced correctly.

If two or more items of set i are stored in row j, they are summed up and the intermediate result is written to $Buf_{red}[j\alpha]$ before $l = \alpha$. Otherwise, the set contains more than $\alpha - 1$ items in row j, and thus contains more than α items in Buf_{in}. Since $Buf_{red}[(j-1)\alpha+\alpha-1]$ is added with $Buf_{red}[j\alpha]$ when $l = \alpha$, the set is reduced correctly.

The case in which set i is split between two buffers is similar to the case where the set is split between row 0 and row $0-1$. Thus, we can prove the set is reduced correctly using a similar analysis as above.

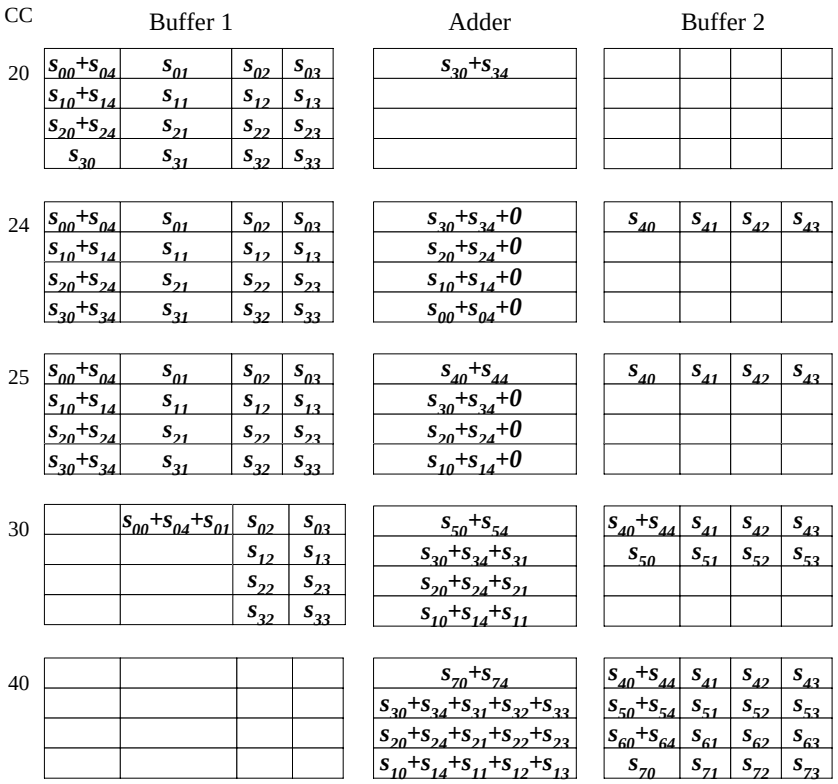
We now prove that r_i only contains items of set i. Suppose during some clock cycle, the adder accepts two operands from different sets. This cannot happen when $l < \alpha$ because it is not allowed by the algorithm. Suppose when $l = \alpha$, the items in $Buf_{red}[(j-1)\alpha + l - 1]$ and $Buf_{red}[j\alpha]$ are from different sets. In this case, no set is split between row $j-1$ and row j; hence these items will not be read by the adder. Similarly, when $l = \alpha$, if the items in $Buf_{red}[0]$ and in *temp* are from different sets, they will not be read by the adder. We thus reach a contradiction. ■

Theorem 6 *The circuit reduces p sets in at most $(\sum_{i=0}^{p-1} n_i + 2\alpha^2)$ cycles.*

Proof. Since the reduction circuit never stalls reading the inputs, the last element of the last set enters the circuit in clock cycle $\sum_{i=0}^{p-1} n_i$.

At this time, if Buf_{in} is full and Buf_{red} is empty, they are swapped. Hence the adder reads from Buf_{red}, which is full. This takes α^2 clock cycles.

If only q ($q < \alpha^2$) entries of Buf_{in} is written as the last input enters, Buf_{red} is not empty. Thus, the adder first reads from Buf_{red} before the two buffers are swapped. This takes $\alpha^2 - q$ clock cycles. Then Buf_{in} becomes Buf_{red} and is read by the adder. The operands of the last addition enter the adder in clock cycle $\sum_{i=0}^{p-1} n_i + (\alpha^2 - q) + \alpha(\alpha-1) + \lceil \frac{q}{\alpha} \rceil$. As $q > \lceil \frac{q}{\alpha} \rceil$, the final result of the pth set is available in



Clock cycle	Notes
20	s_{34} enters the circuit and is added with s_{30}. Since Buffer 1 is full and s_{34} is the last input of set 3, Buf_{in} = Buffer 2 and Buf_{red} = Buffer 1.
	In the next 4 clock cycles, Buffer 1 is read while new inputs are written into Buffer 2.
24	The result of $s_{30} + s_{34}$ is written into Buffer 1.
25	s_{44} enters the circuit and is added with s_{40}; during this clock cycle, Buffer 1 is not read by the adder.
	The algorithm continues like this until Buffer 1 is empty. Then sets in Buffer 2 are reduced.

Figure 4. Snapshots for the basic case

clock cycle $\sum_{i=0}^{p-1} n_i + (\alpha^2 - q) + \alpha(\alpha - 1) + q + \alpha = \sum_{i=0}^{p-1} n_i + 2\alpha^2$. ■

3.4 Illustrative Example

We illustrate the operations of the reduction circuit through examples. Suppose $\alpha = 4$. During clock cycle 0, Buf_{in} = Buffer 1 and Buf_{red} = Buffer 2; both buffers are empty. The inputs are stored in the buffers in row-major order. In particular, the memory addresses increase from left to right, and from top to bottom. The data in the adder travels from the top to the bottom.

A. Basic Case An example for the basic case is shown in Figure 4. There are 8 input sets of size 5. Due to space limitation, we only show the data in the buffers and the adder in several selected clock cycles.

B. General Case For the general case, there are 10 input sets of size 3. We show the data in the buffers and the adders in the selected clock cycles in Figure 5.

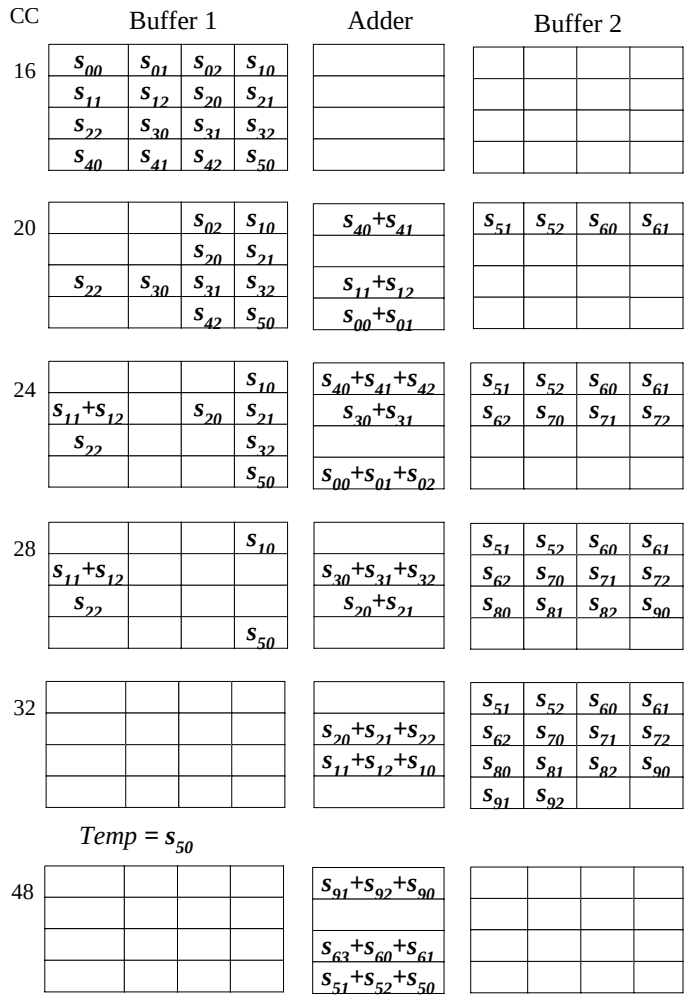
4 Experimental Results

We have implemented our designs on a Xilinx Virtex-II Pro XC2VP7 FPGA. We used Xilinx ISE 7.1i [13] and Mentor Graphics Modelsim 6.0a [6] development tools.

In our experiments, we used our own IEEE-format floating-point adders which are described in [3]. Both the adders are pipelined. Their characteristics are shown in Table 1.

The buffers in the circuit are implemented using the Block RAMs (BRAMs) on the FPGA device. Each Virtex-II Pro BRAM is an 18 Kb true dual-port RAM with two independently controlled synchronous ports that access a common storage area. Since the BRAMs are embedded hardware on the device, they barely increase the total area (the number of slices) occupied by our designs. The characteristics of the 32-bit and 64-bit reduction circuits are shown in Table 2.

We see that the achievable clock speed of the circuit is determined by that of the floating-point adder. On the other hand, the area of the reduction circuit is much larger than that of the adder, due to the control logic and the routing. Since the control in the general case is more complex, the reduction circuit occupies more area. Moreover, in the general case, as the final results of the input sets may be generated out of order, each input carries with it the index of the set it belongs to. Therefore, an additional buffer is needed to store the set indices, and hence more BRAMs are used.



Clock cycle	Notes
16	Buffer 1 is full, and we swap Buf_{in} and Buf_{red}.
	In the next 4 clock cycles, the adder reads operands from 4 rows of Buffer 1. Since s_{22} is not in the same set as s_{30}, they are not added.
	In the next 4 clock cycles, the adder continues reading from Buffer 1. $s_{11}+s_{12}$ is written to the first location in row 1 of Buffer 1.
	Starting from clock cycle 29, the intermediate results of split rows are summed up.
	When Buffer 1 is empty, the adder begins to reduce sets in Buffer 2.
48	The operands of the last addition enter the adder.

Figure 5. Snapshots for the general case

Table 1. Floating-point adders

	32-bit	64-bit
Pipeline Delay	18	14
Area(Slices)	584	892
Clock Speed(MHz)	200	170

Table 2. Reduction circuits

	Basic Case		General Case	
	32-bit	64-bit	32-bit	64-bit
Area(Slices)	923	1506	1017	1658
Clock Speed(MHz)	200	170	200	170
No. of BRAMs	2	4	4	6

4.1 Applications

In [17], we proposed an FPGA-based design for Sparse Matrix-Vector Multiply (SpMXV), $\mathbf{y} = A\mathbf{x}$. The tree-based architecture is shown in Figure 6.

The nonzero elements in each row are partitioned into sub-rows of size k. During each clock cycle, k values in a sub-row enters the multipliers. The resulting k multiply results are reduced by $\lg(k)$ levels of pipelined adders. After the pipeline in the architecture is full, the adder tree yields the result of a sub-row during each clock cycle. Thus, a reduction circuit is needed in the architecture to accumulate these intermediate results.

In the experiments of [17], we used a reduction circuit with $\lg(n)$ adders, where n is the maximum input size. In [17], the circuit contains 7 floating-point adders, and the total delay of reducing p input sets is $\sum_{i=0}^{p-1} n_i + 7\alpha$. The MFLOPS performance of the SpMXV design was calculated as:

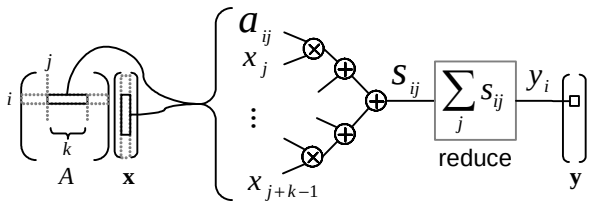


Figure 6. Matrix-Vector Multiply

$$\frac{\textit{total number of floating-point operations performed}}{\textit{total delay of the design}}$$

We evaluated the design using matrices from University of Florida Sparse Matrix Collection [12]. The design achieved more than 350 MFLOPS for every sparse matrix we tested [17].

With the reduction circuits proposed in this paper, the size of the design in [17] will be greatly reduced without any modification to any other part of the design. On the other hand, the total delay for reducing p input sets increases to $\sum_{i=0}^{p-1} n_i + 2\alpha^2$. In practice, α is under 20 and $\sum_{i=0}^{p-1} n_i$ is over tens of thousand. Thus, the SpMXV design achieves almost the same MFLOPS performance with much smaller area.

Many other applications need reduction circuits. In [15], a reduction circuit is used in FPGA-based designs for BLAS (Basic Linear Array Subprograms) operations, including dot product and matrix-vector multiply. In [9], a reduction circuit is needed to perform the accumulation of all three force computations in molecular dynamics. In these applications, the size of the input set is usually much larger than α. Thus, the proposed reduction circuit for the basic case suffices to perform the reduction.

5 Conclusion

We have presented a high-performance and area-efficient FPGA-based reduction circuit for reducing multiple sets of sequentially delivered floating-point values. Unlike the previous work in this area, this circuit only uses one floating-point adder and can handle input sets of arbitrary sizes. Moreover, the proposed reduction circuit only needs two buffers of fixed size α^2, where α is the pipeline delay of the adder. We have implemented the reduction circuit on a Xilinx Virtex-II Pro XC2VP7 FPGA. The size of the circuit as well as its achievable clock speed are independent of the size of each set and the number of the input sets. The compact area and high clock-rate make the circuit an attractive building block for a number of applications which require reduction of series of floating-point values.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.

[2] Cray Inc. Cray XD1™. `http://www.cray.com/products/xd1/`.

[3] G. Govindu, R. Scrofano, and V. K. Prasanna. A Library of Parameterizable Floating-Point Cores for FPGAs and Their Application to Scientific Computing. In *Proc. of International Conference on Engineering Reconfigurable Systems and Algorithms*, June 2005.

[4] P. M. Kogge. *The Architecture of Pipelined Computers*. Hemisphere Pub. Corp., 1981.

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

[6] Mentor Graphics Corp. `http://www.mentor.com/`.

[7] G. R. Morris, L. Zhuo, and V. K. Prasanna. High-Performance FPGA-Based General Reduction Methods. In *Proc. of 10th IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2005.

[8] L. M. Ni and K. Hwang. Vector Reduction Methods for Arithmetic Pipelines. In *Proc. of the 6th International Symposium on Computer Arithmetic*, pages 144–150, June 1983.

[9] R. Scrofano and V. K. Prasanna. Computing Lennard-Jones Potentials and Forces with Reconfigurable Hardware. In *Proc. of International Conference on Engineering of Reconfigurable Systems and Algorithms*, June 2004.

[10] SRC Computers. MAPstation™. `http://www.srccomp.com/MAPstations.htm`.

[11] K. D. Underwood and K. S. Hemmert. Closing the Gap: CPU and FPGA Trends in Sustainable Floating-Point BLAS Performance. In *Proc. of 2004 IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2004.

[12] University of Florida Sparse Matrix Collection. `http://www.cise.ufl.edu/research/sparse/matrices/`.

[13] Xilinx Incorporated. `http://www.xilinx.com`.

[14] L. Zhuo, G. R. Morris, and V. K. Prasanna. Designing Scalable FPGA-Based Reduction Circuits Using Pipelined Floating-Point Cores. In *Proc. of the 12th Reconfigurable Architectures Workshop*, April 2005.

[15] L. Zhuo and V. K. Prasanna. Design Tradeoffs for BLAS Operations on Reconfigurable Hardware. Proc. of the 34th International Conference on Parallel Processing, June 2005.

[16] L. Zhuo and V. K. Prasanna. High Performance Linear Algebra Operations on Reconfigurable Systems. Proc. of the SuperComputing 2005 (to appear), November 2005.

[17] L. Zhuo and V. K. Prasanna. Sparse Matrix-Vector Multiplication on FPGAs. In *Proc. of the 13th ACM International Symposium on Field-Programmable Gate Arrays*, February 2005.