Accelerating Frequent Item Counting with FPGA

Yuliang Sun[1], Zilong Wang[1], Sitao Huang[1], Lanjun Wang[2],
Yu Wang[1], Rong Luo[1], Huazhong Yang[1]
[1]E.E. Dept., TNLIST, Tsinghua University; [2]IBM Research-China
{sunyuliang13, wang-zl11, hst10}@mails.tsinghua.edu.cn, {wangljbj}@cn.ibm.com,
{yu-wang, luorong, yanghz}@mail.tsinghua.edu.cn

ABSTRACT
Frequent item counting is one of the most important operations in time series data mining algorithms, and the space saving algorithm is a widely used approach to solving this problem. With the rapid rising of data input speeds, the most challenging problem in frequent item counting is to meet the requirement of wire-speed processing. In this paper, we propose a streaming oriented PE-ring framework on FPGA for counting frequent items. Compared with the best existing FPGA implementation, our basic PE-ring framework saves 50% lookup table resources cost and achieves the same throughput in a more scalable way. Furthermore, we adopt SIMD-like cascaded filter for further performance improvements, which outperforms the previous work by up to 3.24 times in some data distributions.

Categories and Subject Descriptors
B.7.1 [Integrated Circuits]: Types and Design Styles – Algorithms implemented in hardware

H.2.8 [Database Management]: Database application – Data mining

General Terms
Algorithms, Design, Performance.

Keywords
Frequent item counting, FPGA, time series

1. INTRODUCTION
Frequent item counting focuses on the operation to pick up items which occur most frequently in a given sequence. It is one of the most basic and important operations in many domains, such as data mining [1] and natural language processing [2]. A wide variety of operations in practical applications can be abstracted into this problem, for example, finding the most popular destinations on the Internet [3]. Moreover, the frequent item counting provides intermediate results or acts as a 'subroutine' for many streaming applications, such as frequent item set mining [4]. With the rapid rising of data input speeds, it is important to find a method to implement an efficient and scalable frequent item counting system.

In recent years, researchers have tried parallel multi-core or computer-cluster frameworks for frequent item counting. For example, Roy et al. proposed a sequence of 'cascaded filters' in the preprocessing of skew input data, and made the final core only face a small volume of data [5]. Roy et al. considered a kind of skew data distribution as known as Zipfian distribution, which is widely used to describe many types of data studied in the physical and social sciences. However, their framework loses effectiveness for general data distributions. Moreover, it suffers from the memory wall problem of von Neumann architecture, which causes a time, power and resource waste of CPUs.

Meanwhile, the existing best field programmable gate arrays (FPGAs) implementation [6] achieves a data-independent result, which is 4 times faster than the best software. However, the resource consumption increases linearly with the problem size (i.e., *k* in the top-*k* query), which may cause the scalability problem. The reason is that all the counting results are stored in the registers rather than the block RAMs in FPGA. The existing implementation cannot support a large top-*k* query, especially when *k* is larger than 1000. In addition, the utilization ratio of their processing elements (PEs) is only 50%, which means there is quite a waste of resources.

In this paper, an FPGA framework is proposed for frequent item counting to reduce resources cost while maintaining the performance. Moreover, the optimizations of cascaded filter and SIMD-like method are applied to the basic framework for further performance improvements. The major contributions of our study are as follows:

1) Our proposed PE-ring framework for frequent item counting fixes the positions of the input tuples, while transfers parameters and temporary counting results to achieve scalability. Compared with the previous framework, the PE-ring framework reduces 50% of lookup table resources cost with similar performance.
2) We combine the advantages of cascaded filter and SIMD-like method to improve the performance. The throughput of improved framework is 242 million tuples per second in some data distributions (for 1024 bins whose α is larger than 2 in Zipfian distribution).

2. FRAMEWORK
2.1 Space Saving Algorithm and its FPGA Implementations
The frequent item counting problem is defined as follows. Assume a stream S of size N is drawn from an alphabet A, the Φ-frequent items are those tuples x that occur at least ΦN times in S ($\Phi \in (0,1)$).The number of counting for a tuple x in S is termed the frequency f_x.

If the frequent item counting requires absolute accuracy, the occurrences of all tuples have to be counted in memory. This resource consumption would be totally unrealistic. Usually, practical applications can tolerate a small error ε in the counting result. The counting result may include some additional tuples for which $(\Phi\text{-}\varepsilon)N < f_x < \Phi N$. The space saving algorithm is proposed to meet the requirement of limit space consumption [7]. In the space saving algorithm, *k* bins are set to only count the most frequent tuples in stream *S*. Each bin is composed of two parameters: item and count. To achieve an error no more than ε, *k* should be larger than *1/ε*.

```
 1: for each $x$ in Stream $S$ do
 2:     for $i = 1 : k$ do
 3:         if $b_i.item == x$ then
 4:             $b_i.count = b_i.count + 1$;
 5:             $x = Null$;
 6:         end if
 7:         if $b_i.count > b_{i-1}.count$ then
 8:             swap( $b_{i-1}$, $b_i$ );
 9:         end if
10:     end for
11:     if $x \neq Null$ then
12:         $b_k.item = x$;
13:         $b_k.count = b_k.count + 1$;
14:     end if
15: end for
```

Figure 1 Space saving customized for PE-ring

The space saving algorithm customized for PE-ring is shown in Figure 1. In step 1, the input tuple *x* is compared with *bin$_i$.item*. Corresponding addition operation is executed if they match (line 3~6). In step 2, the bin swaps its contents with its neighbor if their counts are reversed (line 7~9), which is similar as the bubble sort algorithm. The next bin is pushed in after all these operations are done in step 3 (line 2). In addition, the last bin will be replaced if there is no bin matches the tuple (line 11~14).

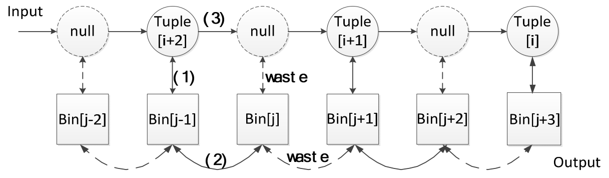


Figure 2 Pipelining array on FPGA [6]

A previous framework for FPGA based frequent item counting is shown in Figure 2 [6]. Bins are arranged by their count. The tuple is the current input data that waits to be considered, and each of them needs to be compared with the bins in order. For every input tuple, a counter is allocated. Along with the flowing from the first counter to the last one, an increment operation is triggered if the tuple matches the item of the current bin (step 1). If the adjacent bins are not in order, their contents are exchanged (step 2). Another delivery of tuples starts when all these operations are done (step 3). When matching and swapping are executed in one counter, its neighboring counters are idle, which is a waste of resources.

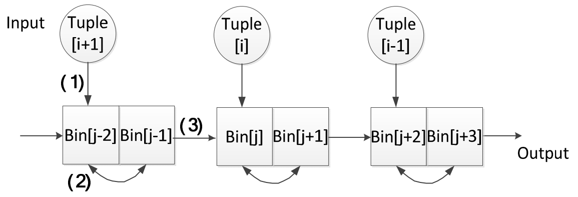


Figure 3 Stream oriented PE-ring on FPGA

A stream oriented PE-ring framework is proposed in which bins rather than input tuples are flowing in Figure 3. Both frameworks accomplish the same function in the space saving algorithm. Matching and swapping operations are fixed in independent elements to avoid redundant resource consumption. The PE-ring framework is adapted to a full utilization of the advantage in fine-grained parallelism on FPGA.

2.2 PE-ring Framework

A PE is the smallest structure executing the operations for one input tuple shown in Figure 4. In each PE, two bins are recorded in the registers. The neighbor input is the bin from the last PE. The global input is a tuple from time series, distributed by a time series router. The neighbor output bin is passed to the next PE.

The operation in each PE corresponds to the algorithm in line 4~14 in Figure 3. Two neighbor bins are recorded in each PE. In every cycle, PE compares the input *tuple* and the item of the input bin$_j$, and the count of the input bin will be accumulated if the input *tuple* matches *bin$_j$.item*. In order to pick up the bin with minimum count value at the end of the bin series, an adjustment of the storage position of two bins in the same PE is executed. Moreover, every bin has a tab to record whether it is the last bin in the bin series. The item of the last bin will be updated by the input *tuple* in case the input tuple has not matched.

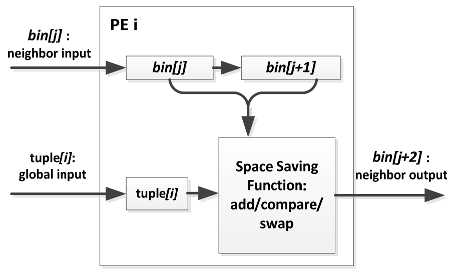


Figure 4 Architecture of PE

This PE structure ensures that in the PE-ring, the storage of input tuple is fixed and the bins are delivered forward to the neighbor PE. The framework exactly matches the operations in the algorithm. In a single PE, one tuple and two bins are handled in every cycle to fully use the resources without any idle time. In Section 4.1, experimental results show that compared with [6], 50% lookup table resources cost is saved for the whole system.

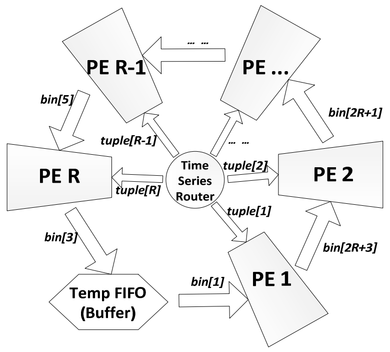


Figure 5 Architecture of PE-ring

Based on the experience from our previous work [8], PEs are linked as a ring in Figure 5, which is called the PE-ring. PEs communicate with their neighbors by the flowing bins in the PE-ring. Moreover, a distribution of a new input tuple is executed through global routing after finishing the comparison of current tuples. According to the mobility of bins in the PE-ring, the sequential output bins from a certain PE are the current accumulated result by the end of the tuple in this PE.

As introduced in Section 2.1, the amount of bins needs to be large enough to achieve high precision in space saving. The number of PEs in a PE-ring is very large and limited by resources on FPGA. To solve this problem, a buffer is inserted to the architecture. The output bin from the last PE is pushed into the buffer for temporary storage, which can be achieved by using a temp FIFO. This bin can be later delivered to the first PE again to start a new circulation if the number of PEs in the PE-ring is less than *k*.

A PE-ring with a proper buffer can save a lot of on-chip resources cost compared with the situation that all the bins are stored in PEs. The temp FIFO uses the space on the block RAMs rather than the registers on FPGAs. Besides, the buffer also guarantees the scalability for PE-ring. The limited on-chip resources can be a strict restriction for the scale of the PE-ring without a buffer. If frequent item counting requires high accuracy, the number of bins would be too large for an FPGA to afford it. Therefore, a buffer in the PE-ring which stores those redundant bins that cannot be put into the PE-ring would be very beneficial.

3. OPTIMATIONS

3.1 Cascaded Filter

In many domains, especially in linguistics [9] and social sciences [10], many types of data are approximated with a Zipfian distribution and can be called skew tuples. Skewness means a measure of the asymmetry of a probability distribution. Skew tuples have a feature that most of the tuples find their matches at the first several bins in space saving. Zipfian distribution is described in the equation below, where N is the number of tuples, r is the tuple's rank in the frequency table of the dataset, and α is the value of the exponent parameter in the distribution. The bigger α is, the skewer this distribution becomes.

$$f(r,\alpha,N)=\frac{1/r^{\alpha}}{\sum_{n=1}^{N}(1/n^{\alpha})} \qquad (1)$$

Cascaded filter is suitable to accelerate frequent item counting for skew tuples. The intention of cascaded filter is to place a filter which picks up those input tuples with high frequencies. By routing those most frequent tuples through a shortcut code path, the workload in the module of space saving can get a significant relief [5].

There are two parts of bins in the cascaded filter strengthened PE-ring. Part I is stored in the cascaded filter and part II on in PE-ring. Part I, which is in a small scale, contains the bins with highest frequencies and corresponds to the shortcut path processing. Part II is used for the exact space saving implementation. Tuples will be sent into PE-ring only after their flowing through the cascaded filter without a match. Especially when the input tuples are skew enough, the cascaded filter is able to catch most of the tuples so that the workload in PE-ring for space saving can be greatly reduced.

In our software simulation of the cascaded filter algorithm, tuples with different α values in Zipfian distribution are tested. Figure 6 shows the ratio that tuples are caught in the filter over streams. If over a half tuples find their matches in the cascaded filter, the workload for the PE-ring is halved. In this scenario, a PE-ring with a same length of buffer can complete this work which saves half resources cost compared with a single PE-ring.

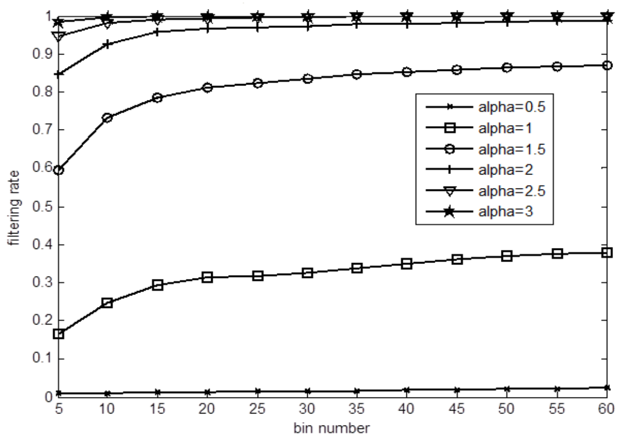
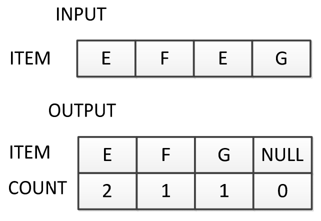


Figure 6 Filtering rate with different number of bins

3.2 SIMD-like method

For the cascaded filter strengthened PE-ring, all the input tuples pass through the cascaded filter. The performance of the cascaded filter determines the performance of the frequent item counting system. A SIMD-like method is applied in the filter to improve the performance. The SIMD-like method is analogous to single instruction multiple data (SIMD) from the multi-core implementation [5]. If *m* tuples are compared with one bin in every cycle, then the throughput of the whole system can be about *m* times over before. The SIMD-like method focuses on the step of comparison between tuples and bins and tries to parallelize this step to speed up.


INPUT

ITEM	E	F	E	G

OUTPUT

ITEM	E	F	G	NULL
COUNT	2	1	1	0

Figure 7 Data collation for SIMD

As shown in Figure 7, the same tuples should be aggregated together before the cascaded filter. To make such collation or not is crucial for the circuit structure design. If the tuple collation is not implemented, uncertain times of accumulation have to be considered in PE and *m* counters are required. The circuit structure in each PE would become more complex without the tuple collation.

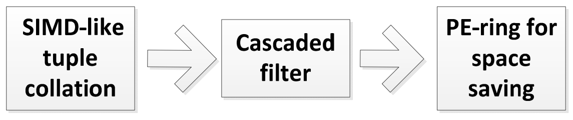


Figure 8 Framework for SIMD-like cascaded filter

The overall framework for PE-ring combined with cascaded filter and SIMD-like method is shown in Figure 8. After coming in, the input tuples move through three modules: tuple collation, cascaded filter and PE-ring. Based on the scalability of PE-ring, the cascaded filter and SIMD-like method are inferred for further performance improvements for data with skew distributions.

4. EXPERIMENT

We choose the platform Virtex-6 XC6VLX550T to make rational comparison with the previous work that uses the same device [6]. The word width of tuples and counting results are 32 bits.

4.1 PE-ring Framework

Table 1 Pipelining array [6] and PE-ring implementation

Bins	LUT		Register		Clock (MHz)		Throughput (M items/s)	
	array	PE-ring	array	PE-ring	array	PE-ring	array	PE-ring
64	4%	2%	1%	1%	110	267	110	133
128	9%	4%	1%	1%	115	255	115	127
256	20%	7%	3%	3%	105	246	105	123
512	39%	15%	6%	6%	110	217	110	108
1024	78%	31%	12%	12%	95	224	95	112
2048	-	64%		24%	-	202	-	101

Table 1 shows the results of the previous work [6] and the PE-ring framework. For example, for 256 bins, both of these two methods uses 3% register. However, our method only uses 7% LUTs, which is less than half of previous one. Meanwhile, it is note that PE-ring processes one tuple in every two cycles, so that our throughputs are over 100M items/s, which are comparable to the previous work. That is to say, PE-ring achieves a similar performance with reducing 50% of lookup table. Furthermore, since we balance the utilization of registers and LUTs, our PE-ring achieves 2048 bins, which means a high accuracy in frequent item counting.

4.2 SIMD-like Cascaded Filter Framework

Table 2 PE-ring with FIFO and filter implementation (16 bins in filter)

Bins in PE-ring	LUT	Register	Filter clk (MHz)	PE-ring clk (MHz)	Throughput (M items/s)
256	11%	4%	200	100	400
1024	35%	14%	121	60	242

We implement a SIMD-like cascaded filter to pursuit a higher performance, especially for skew data. We set 4 tuples processed in the cascaded filter at the same time. The input data is with a Zipfian distribution with a parameter α no less than 2. Table 2 shows the results of framework. Comparing with the results from Table 1, for the same precision (bin size), the throughput of improved framework outperforms the basic framework. For example, as 256 bins, the optimized method achieves a throughput as 400M items/s which is more than three times of the basic implementation.

In addition, it is note that the existing per-line bandwidth of Peripheral Component Interconnect Express (PCIe) is 1 GB/s. The throughput here requires the memory bandwidth from off-chip is 968 MB/s, almost approaching the bandwidth limitation.

5. CONCLUSION

In this paper, we propose a novel FPGA framework for frequent item counting to achieve scalability and save 50% lookup table resources cost. Unlike the traditional framework that makes input tuples flowing through, our PE-ring structure passes through parameters and operations to implement space saving algorithm. In addition, we leverage the idea of cascaded filter and SIMD-like method to achieve higher performance for skew input tuples.

6. ACKNOWLEDGMENTS

This work was supported by IBM Research China University Relationship Program, National Natural Science Foundation of China (No. 61373026, No. 61261160501, No. 61028006), 973 project 2013CB329000, National Science and Technology Major Project (2013ZX03003013-003), and Youth Talent Development Plan of Beijing (YETP0099).

7. REFERENCES

[1] Liu J, Pan Y, Wang K, et al. Mining frequent item sets by opportunistic projection. Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2002: 229-238.

[2] Beil F, Ester M, Xu X. Frequent term-based text clustering. Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2002: 436-442.

[3] Cormode G, Hadjieleftheriou M. Finding frequent items in data streams. Proceedings of the VLDB Endowment, 2008, 1(2): 1530-1541.

[4] Chakrabarti A, Cormode G, McGregor A. A near-optimal algorithm for computing the entropy of a stream. Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, 2007: 328-335.

[5] Roy P, Teubner J, Alonso G. Efficient frequent item counting in multi-core hardware Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2012: 1451-1459.

[6] Teubner J, Muller R, Alonso G. Frequent item computation on a chip. Knowledge and Data Engineering, IEEE Transactions on, 2011, 23(8): 1169-1181.

[7] Metwally A, Agrawal D, Abbadi A E. An integrated efficient solution for computing frequent and top-k elements in data streams. ACM Transactions on Database Systems (TODS), 2006, 31(3): 1095-1133.

[8] Wang Z, Huang S, Wang L, et al. Accelerating subsequence similarity search based on dynamic time warping distance with FPGA. Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays. ACM, 2013: 53-62.

[9] Montemurro M A. Beyond the Zipf–Mandelbrot law in quantitative linguistics. Physica A: Statistical Mechanics and its Applications, 2001, 300(3): 567-578.

[10] Gabaix X. Zipf's Law and the Growth of Cities. The American Economic Review, 1999, 89(2): 129-132.