2018 21st Euromicro Conference on Digital System Design

Heavy-hitter detection using a hardware sketch with the Countmin-CU algorithm

Antonio Saavedra*, Cecilia Hernández†‡, and Miguel Figueroa*
*Department of Electrical Engineering, †Department of Computer Science - Universidad de Concepción, Concepción, Chile
‡Center of Biotechnology and Bioengineering (CeBiB)
e-mail: {antsaavedra, cecihernandez, miguel.figueroa}@udec.cl

***Abstract*—We present a custom hardware architecture for fast heavy hitter detection in large data streams. The architecture probabilistically estimates the frequency of each element in the data stream using the Countmin-CU sketch with the H3 family of hash functions. The sketch is stored in on-chip memory, and the architecture exploits the parallelism available in the data by simultaneously processing each row of the sketch. The hash functions map each element to a set of counters on the sketch, and the sketch increments the counters that hold the minimum value, which corresponds to the estimated frequency of the element. The hash functions and sorting network are implemented in hardware as fully pipelined circuits, in order to maximize their operating clock frequency. We show a prototype of the architecture running on a Xilinx Kintex-7 XC7K325T FPGA operating with a 300MHz clock, which can process a stream of 3,982,496 32-bit elements and detect the heavy hitters with an 4×16,384-element sketch in 13.27ms, achieving a speedup of 768 compared to a modern desktop computer.**

Keywords*—Heavy hitter; streaming algorithm; hardware acceleration; Field-Programmable Gate Array

I. Introduction

The *heavy hitter* problem is one of the most studied in the data stream model and has been used as a building block in many applications. It consists of finding the most frequent elements above a user-supplied threshold in a stream. Assuming a large stream, the problem of estimating the most frequent elements is not simple, since it is unfeasible to keep a counter for each distinct element in memory and provide fast access to each counter. Considering that many applications need to identify only the most frequent elements, stream algorithms aim to estimate heavy hitters and their frequencies using sublinear space and fast processing time in one pass over the stream. Computing heavy hitters has many applications such as network traffic analysis [1], k-mer counting in bioinformatics [2], and social networks analysis [3].

Network traffic analytics include tasks such as discovering Denial of Service (DoS) attacks, identifying heavy network users, diagnose performance problems and balancing traffic load. Finding heavy hitters in network traffic has been traditionally used for analyzing packet samples or flow logs. More recently, it has been used for aggregating traffic statistics and identifying large flows in real time directly in the data plane [4], [1]. Because commodity network switches have limited resources, estimating heavy hitters with high speed, accuracy and small memory is important to avoid adding latencies and enabling switches to operate with high throughput [1].

In bioinfomatics, an important challenge in next-generation genome sequencing (NGS) is to assemble massive overlapping short reads that are sampled from DNA fragments. A fundamental task in many assembly algorithms is estimating the frequency of k-mers, where a k-mer is a subsequence of k bases in DNA sequences. Efficient k-mer counting plays an important role in different processes, including data preprocessing for novo assembly, repeat detection, and sequence coverage estimation [5]. K-mer counting has also been used in motif discovery. A DNA motif is a sequence pattern with biological significance and motif prediction models aim to identify transcription factor binding sites or motifs in Chip-seq data [6]. In recent years, several k-mer counting approaches have been proposed to deal with memory consumption and processing time [7].

Fast heavy-hitter detection in these applications requires high computational throughput that is often above the capabilities of traditional computers. Current hardware technologies such as Field Programmable Gate Arrays (FPGAs) and Graphics Processing Units (GPUs) are able to efficiently exploit the large-scale and fine-grained data parallelism present in these applications to achieve high performance at moderate cost. Therefore, a hardware accelerator for heavy-hitter detection could potentially enable the application to meet demanding real-time constraints, and greatly reduce the execution time of data processing and analysis of large datasets.

Methods for solving the heavy hitters problem include deterministic models such as counter-based algorithms, and statistical and probabilistic models such as sampling and sketching algorithms [8]. Counter-based algorithms require dynamic data structures and frequently query the stored elements. Due to their complexity, these techniques are ill-suited for hardware acceleration. On the other hand, sketch-based algorithms use fixed data structures and directly update them using increment/decrement operations. Their limited memory requirements, high level of parallelism, and simplicity of the update and estimation operations, make these techniques much better suited for acceleration using special-purpose hardware.

In this paper, we propose a special-purpose architecture for heavy hitter detection that uses the Countmin-CU sketch algorithm [9], and test its performance on network traffic and Chip-seq datasets. We show experimental results on a Kintex-7 FPGA, which implements a $4 \times 16,384$-counter sketch using on-chip memory. Our design achieves a throughput of up to one data element per clock cycle, with an initial latency of 25 cycles and a clock frequency of 300MHz.

978-1-5386-7377-5/18/$31.00 ©2018 IEEE
DOI 10.1109/DSD.2018.00022

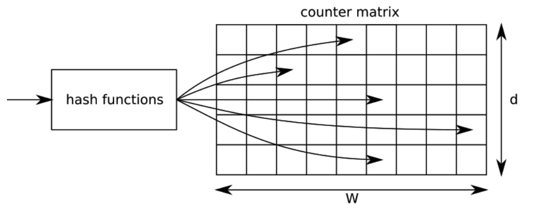


Fig. 1. Generic sketch structure.

II. Related work

Motivated by the increasing need of many applications to deal with massive data, there has been a great attention to streaming algorithms. Algorithms for computing functions in the data stream model need to satisfy a small space requirement, at most poly-logarithmic in the stream size, fast update and query processing, and high accuracy.

Typical sketch-based algorithms use $d \times W$ counters in memory. This can be seen as a sketch with d rows of W counters, as shown in Fig. 1. Sketch algorithms use d hash functions to map each input elements to one counter in each row of the sketch for inserting and estimating frequencies. Two of the most used sketch-based algorithms for estimating heavy hitters are CountSketch [10] and Countmin [11]. CountSketch (CS) uses a pair of hash functions for updating and estimating frequencies in the sketch. One hash function maps the input element to a counter in a row and the other determines whether to increment or decrement the counter. The estimated frequency of an element is obtained by finding the median among corresponding counters. The error is bounded by $\epsilon||a||_2$ with a probability of $1-\delta$ in a space of $\mathcal{O}(\frac{1}{\epsilon^2} \times \log(\frac{1}{\delta}))$.

The Countmin (CM) [11] sketch uses only one hash per row, which is used to map input elements to a counter in a row. The update process always increments counters of an input element. The estimated frequency of an element is obtained by taking the minimum of the counters associated to the element. The CM sketch estimation error is bounded by $\epsilon||a||_1$ with a probability of $1-\delta$ in a space of $\mathcal{O}(\frac{1}{\epsilon} \times \ln(\frac{1}{\delta}))$. The Countmin-CU (CM-CU) [9] sketch operates similarly to CM, but only increments the counters associated to the input element that store the smallest value.

There are several hardware implementations for estimating heavy hitters based on sketch algorithms [12], [13], [14], [1]. For instance, the CM sketch was implemented in hardware using an FPGA for network traffic analysis [12]. The architecture was prototyped on a Xilinx Virtex-7 XC7VX1140 FPGA with a 5×64K-counter sketch. Another work [13] also uses the CM sketch implemented on a FPGA for network analysis. It performs feature extraction with a 4×16K sketch on a Xilinx Virtex II XC2V1000 FPGA with a 270MHz clock. Recently published work [14] proposes an FPGA implementation for heavy hitters applied to network traffic analysis. This approach implements the CS sketch using a Xilinx Virtex UltraScale FPGA, achieving online heavy hitter detection for 100 Gbits/s Ethernet links in a 8×1M-counter sketch. Another hardware implementation proposes a generic fully-pipelined architecture to accelerate CM and K-ary sketches [1]. This work includes an optimization for sketch post-processing techniques for heavy hitter and heavy change detection. The best performance uses a sketch of 5×32K counters. The hardware perform network analysis achieving a throughput of 150 Gbps for heavy-hitter detection and 100Gbps for heavy change detection.

III. Algorithm analisys

In this section, we define and analyze the CM-CU sketch algorithm for estimating heavy hitters. A heavy hitter is formally defined as follows:

Definition 1. Heavy hitter
Given a stream $\mathcal{S}$ and a threshold Φ, a *heavy hitter* (HH) is an element whose frequency in $\mathcal{S}$ is no smaller than Φ. If f_e denotes the frequency of element e in $\mathcal{S}$. Then HH $= \{e|f_e \geq \Phi\}$.

A. Countmin-CU sketch algorithm

The CM-CU algorithm uses the sketch shown in Fig 1 with a family of hash functions h. The sketch is initialized with every counter set to zero as shown in Algorithm 1.

Algorithm 1 Countmin-CU initialization algorithm

```
Require: $d$ hash functions of $W$ counters
1: $C[1,1]...C[d,W] = 0$
2: for $j = 1$ to $d$ do
3:     Initialize hash function $h_j$
4: end for
5: return sketch $C(d, W)$
```

The sketch is updated for each new element of the input stream. The input element is distributed onto a counter in each row by its corresponding hash function. These corresponding counters are then incremented only when a counter holds the minimum value among the d selected counters, which corresponds to the estimated frequency of the element at this time. This is a modification to the traditional CM sketch, which always increments counters for every update no matter their counter values. The update algorithm for the CM-CU sketch is shown in Algorithm 2.

Algorithm 2 Countmin-CU update algorithm

```
Require: sketch $C(d, W)$, input element $a_i$.
1: for $j = 1$ to $d$ do
2:     $min\_est$ = Estimate$(C, a_i)$
3:     if $C[j, h_j(a_i)] == min\_est$ then
4:         $C[j, h_j(a_i)] = C[j, h_j(a_i)] + 1$
5:     end if
6: end for
```

For estimating the frequency on the stream of a given element, Algorithm 3 is used. The minimum value among all d corresponding counters is taken. This requires knowing all the counter values associated with an element by calculating each of the hash functions, and then computing the global minumum. This process is the same as CM.

The advantage of the conservative update of CM-CU is that it mitigates part of the overestimation error induced by collisions with low frequency elements. Nevertheless, it has an impact on the hardware architecture that is discussed in Section IV because the algorithm must estimate the frequency of the element before each update.

Algorithm 3 Countmin-CU estimate algorithm

```
Require: sketch $C(d, W)$ and input element $a_i$.
1: return $\min\{C[j, h_j(a_i)]\}, j \in [1, d]$
```

B. Hash functions

In order to distribute the input elements randomly on the counters and avoid collisions, sketch algorithms generally require hash functions that are universal or universal-2.

A family $\mathscr{H}$ of hash functions is defined as universal if it is constituted by a set of functions from $\mathscr{U} \mapsto w$ and that the expected number of collisions is bound as follows:

For any $x, y \in \mathscr{U}$,

$$Pr_{h \in \mathscr{H}}\left[h(x) = h(y)\right] \leq \frac{1}{w} \tag{1}$$

In other words, any two keys of the universe collide with probability at most $1/w$ when the hash function h is drawn randomly from $\mathscr{H}$. Furthermore, a family of hash functions $\mathscr{U} \mapsto w$ is universal-2 if, additionally, with $x, y \in \mathscr{U}$, for every $s, t \in [w]$, it satisfies

$$Pr_{h \in \mathscr{H}}\left[h(x) = s \wedge h(y) = t\right] = \frac{1}{w^2} \tag{2}$$

Due to the direct implementation possibilities in hardware[15], the selected hash families are the H3[16] hash functions. This is a universal-2 family, defined as follows:

Given a space $\mathscr{U}$ of input keys of n bits and a space w of seeds of m bits, we call Q the set of all the possible $n \times m$ boolean matrices. For a given $q \in Q$, we call $q(i)$ its i-th row, and for an input key x, we call $x(i)$ its i-th bit.

The function $h_q(x) : U \mapsto w$ is then defined as:

$$h_q(x) = x(1) \cdot q(1) \oplus x(2) \cdot q(2) \oplus ... \oplus x(n) \cdot q(n) \tag{3}$$

where $x(i) \cdot q(i)$ denotes the logic AND between the bit $x(i)$ and every bit of $q(i)$, and $\oplus$ is the bitwise logic XOR operation between two bit vectors.

The set $\{h_q | q \in Q\}$ is defined as the H3 family. This family can be efficiently implemented in hardware, because it only uses bitwise AND and XOR operations that can be directly mapped to logic circuits.

IV. System architecture

Fig. 2 shows a block diagram of the hardware architecture used to implement the CM-CU sketch algorithm. The architecture uses internal blockRAM to implement the matrix that stores the counters, with parallel access to each row of the sketch. It also features a high-pipelined implementation of the hash functions and counter update operations, which operate in parallel for each row. Our implementation is fully configurable through a script that automatically generates the sketch based in user-supplied parameters. In particular, the experimental results presented in Section V are based on an implementation that uses a 4-row matrix of 16,384 (16K) counters each, operates on 32-bit elements of the data stream, and runs with a 300MHz clock.

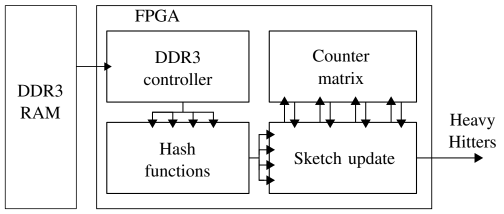


Fig. 2. Hardware architecture of the Countmin-CU sketch.

The main processing blocks of the architecture are:

- A hardware implementation of the H3 hash functions, which are used to select a counter to update in each row of the array.
- The sketch update logic, which implements the read/write interface with the array and computes the counter update operations for each row. It selects the minimum among the counters addressed by the hash functions, and increments its value.

Other important blocks include:

- The DDR3 RAM controller, which accesses the external memory that stores the data stream, and implements a local buffer that allows the sketch to access the data.
- The matrix that stores the counters, implemented on-chip blockRAM memory.

The architecture exploits the data parallelism in the algorithm on two levels: Firstly, each row of the sketch is managed independently and in parallel by separate hash functions and memory access circuitry. The matrix is implemented using separate memory blocks for each row, thus allowing parallel counter reads and updates. Secondly, the circuits that compute the hash functions, counter increments, minimum value selection and memory updates are fully pipelined to maximize the system clock frequency. In the rest of this section, we describe the main components of the architecture.

A. Hash functions

Each of the d rows of the sketch implements the H3 hash function shown in Eqn. (3) with a different set of seeds q. The hardware implementation of each function, shown in Fig. 3, uses a multistage logic network to compute the output h. The first stage is a multiplexer array that implements the AND operations between an each bit of the n-bit key x, provided by the current data element, and each of the n m-bit seeds, which are constants chosen for each of the d rows of the sketch before logic synthesis, with $m = \log_2 W$. In the prototype used to obtain our experimental results, $d = 4$, $n = 32$, and $m = 14$, but their value can be easily changed as parameters of the script that automatically generates the array HDL code.

A network of $\log_2 n$ XOR gates reduces the outputs of the multiplexers and produces the m-bit output h. This output

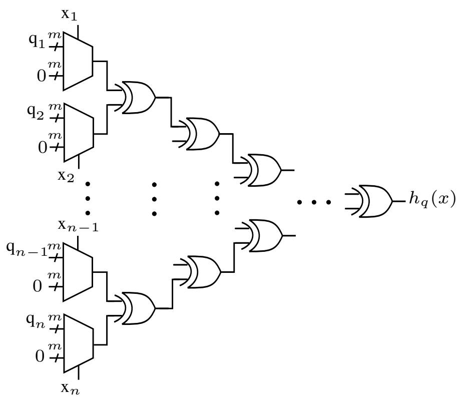


Fig. 3. Hardware implementation of the H3 hash function.

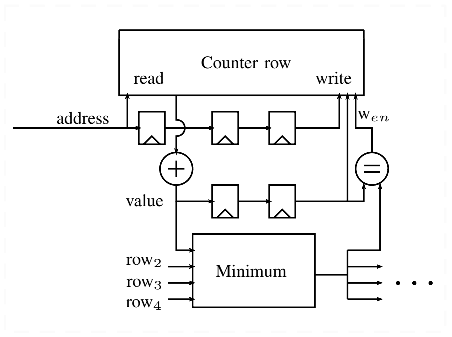


Fig. 4. Sketch update logic

is then used to address the blockRAM memory and select a counter within the corresponding row of the array. The counter values read from each row are then sent to the sketch update logic to compute the new values of the counters. The entire implementation of the circuit is pipelined in $\log_2 n + 1$ (6 in our prototype) stages to maximize clock frequency, although the pipeline registers have been omitted in Fig. 3 for clarity.

B. Sketch updates

The CM-CU algorithm computes the minimum value of the d counters read in each sketch access, and increments the value of only those counter values equal to the minimum. Fig. 4 depicts the hardware responsible for determining which counters must be incremented during each sketch update, and storing their new value. The figure shows the memory access logic for one of the four rows of the sketch in our prototype implementation, and the global minimum computation circuit used to select which rows to update in the sketch.

The module receives the output h_q from each of the d hash function circuits depicted in Fig. 3, and uses it to access the blockRAMs that store each row of the sketch. This operations has a latency of one clock cycle and is performed in all the rows in parallel. Each counter value obtained by the read operation is simultaneously incremented by one, and a the d values are processed by a module that computes the global minimum. This operation has a latency of $\log_2 d$ clock cycles (2 cycles in our implementation). The minimum is then sent

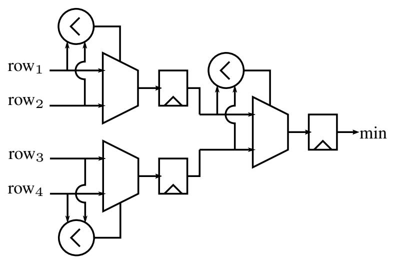


Fig. 5. Minimum computation logic.

to the write port of all d row memories, but the write enable signal is only asserted when the original value is equal to the minimum. The original value and address are delayed by two clock cycles to match the latency of the minimum computation block. The output of the minimum computation circuit is also output as the sketch estimation of the occurrence count of the current input.

Fig. 5 shows the circuit used to compute the global minimum among the d counters, for a sketch of four rows. The circuit is a traditional sorting tree, where only the path that produces the minimum value has been preserved, and the rest of the comparators and multiplexers have been pruned. The circuit has $\log_2 d$ stages, and is fully pipelined to process a new input at each clock cycle.

Because the minimum computation circuit is fully pipelined, a new address is processed on each clock cycle. Unlike CM and CS, the CM-CU algorithm can not update the sketch until the global minimum computation has been finalized. Therefore, the two-cycle latency of the minimum computation circuit introduces a coherency hazard in the pipeline, where it is possible that a new output of the hash function matches one that has not been yet committed its update to memory, and the new access increments and old value of the counter. Eliminating this hazard would require detecting this condition and stalling the pipeline, which complicates the hardware design and potentially degrading the performance of the architecture. However, in order for this hazard to produce a discrepancy in the counter values, the following conditions must occur:

1) A pair of inputs separated by two or fewer clock cycles must produce the same output in at least one hash function.
2) The counters selected by the two conflicting functions must be the minimum of the set in both cases.

If both conditions occur, the algorithm will underestimate the value of the counter by one count. However, this will modify the estimated heavy hitter set only when the modification of the final value of the counter is large enough to cross the threshold used to select the heavy hitters. A theoretical analysis, corroborated by the experimental results presented in Section V-B using real datasets, shows that the hazard produces a negligible error in the frequency estimates, and does not modify the heavy hitter set estimated by the algorithm. Therefore, we do not attempt to detect the hazard in our prototype.

C. Memory interface

As depicted in Fig. 2, our hardware Counmin-CU sketch accesses its input data stream from on-board DDR3 memory. The on-chip DDR3 controller reads the data from the external memory onto an on-chip buffer implemented using blockRAM using an AXI-4 interface. The controller receives the base address and size of the dataset, and it schedules burst reads from the DDR3 memory in order to maximize bandwidth.

The on-chip buffer receives the data from the DDR3 controller, stores it in blockRAM memory, and provides the inputs to the d circuits that implement the H3 hash functions. The schedule of the input transfers to the hash functions depends on the nature of the data (e.g. sequential reads, overlapping strings, selected words). The buffer also compensates for latency differences between external memory and the sketch circuit, thus providing a new set of inputs to the hash function on each clock cycle.

D. Heavy hitter detection

For each new input, CM-CU estimates its current frequency as the minimum of the d counter values selected by the hash functions for that input. Thus, our circuit uses the output of global minimum circuit of Fig.5 as the frequency estimate for the data read from the memory buffer l clock cycles before, where l is the global latency of the pipeline. This estimate is compared against the user-supplied threshold to determine whether the element is a heavy hitter. Heavy hitters are then presented at the output of the circuit, along with their current frequency estimate. Further processing can take place on a host computer or an additional dedicated processor on the chip, and depends on the nature of the application. For example, when analyzing network traffic, a priority queue of size N can be used to select the top N addresses producing traffic on the network [1]; alternatively, in a bioinformatics application, the heavy hitters selected in the input dataset of DNA sequences will be compared against the heavy hitters in a control dataset to select emerging motifs [17].

V. Experimental results

A. Datasets and performance metrics

We developed an optimized software framework to test different sketches. The software is written in the C language and uses various optimizations to maximize its performance running on a single thread. In addition to CM-CU, the framework supports CM and CS in order to compare our results to other published hardware accelerators [1], [14],

We tested the performance of the three sketches using two standard datasets: First, we used network traffic data from the 2016 Equinox-Chicago ISP backbone link traces from the Center for Applied Internet Data Analysis (CAIDA) [18]. We analyzed an stream composed of the parsed source IP addresses from 2,825,605 IPv4 packets from one of the traces. The IPv4 IP addresses are represented in 32 bits. The second dataset is the widely used mESC (mouse embryonic stem cell) Chip-seq database [19]. The test data in this dataset consists of 21,644 peak regions with a length of 200 DNA bases. There are four different bases, represented by the ASCII characters 'A', 'T', 'C', and 'G'. We used the sketch to detect which k-mers of length 16 (a subsequence of 16 bases) are heavy hitters. The algorithm analyzes overlapping strings, therefore each 200-base line contains 184 $(200-16)$ k-mers. Thus, the entire database contains 3,982,496 k-mers. The sketch uses two bits from each ASCII character to represent the four bases, so it treats each k-mer of length 16 as a 32-bit word.

TABLE I. Dataset characterization

Database	Total elements	Distintc elements	Single occurrences
mESC	3,982,496	3,875,839	3,816,248
CAIDA	2,825,605	116,524	34,207

Table I summarizes information of the two datasets. The CAIDA database is smaller than mESC but, more importantly, has much fewer different elements. Indeed, only 4.1% of the elements in CAIDA are distinct, while in mESC 97.3% of the total elements in the set are distinct. As a consequence, 98.5% of the distinct elements in mESC occur only once in the dataset, and this fraction is only 29.4% in CAIDA.

Fig. 6 shows the frequency distribution of the 100 most frequent elements in both datasets. In order to determine a threshold to select the heavy hitters in the mESC database, we used the bioinformatics criterion proposed by Yu [17], which results in a frequency threshold of 145 for a 16-base string in our dataset. For the CAIDA database, we selected a frequency threshold of 6,000 based on the observed distribution of the most frequent elements. Fig. 6 shows also the selected threshold for both datasets.

B. Sketch performance

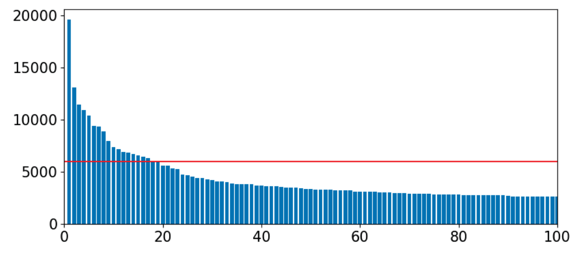
We compare the performance of the three sketches (CS, CM, and CM-CU) with two commonly used metrics for information retrieval: Precision and recall. The precision of an algorithm measures *how much of the information it retrieves* is relevant, and recall measures *how much of the relevant information* it retrieves. In our case, when an element of the dataset is selected as a heavy hitter, it can be either a true positive if a fully-precise deterministic algorithm also selects the element, or a false positive if the deterministic algorithm does not select it. If the sketch algorithm does not select a data element, it can be a true negative if the deterministic algorithm also does not select it, or a false negative if the deterministic algorithm selects it. Thus, precision and recall are defined as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5}$$

where TP is the number of true positives, FP is the number of false positives, and FN is the number of false negatives.

Because CM and CM-CU always increment their counters, they can overestimate the frequency of the elements in the dataset, but they do not underestimate it. In consequence, these two algorithms can produce false positives, but not false negatives, and their recall is always 1. In contrast, CS can increment and decrement counters, and therefore can produce both false positives and false negatives.


(a) CAIDA database

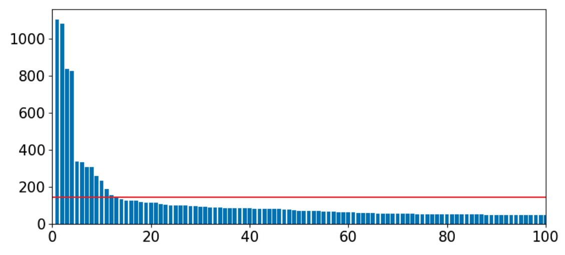

(b) mESC database

Fig. 6. Frequency distribution of the top-100 elements in the CAIDA and ESC databases, and the frequency threshold used to select the heavy hitters.

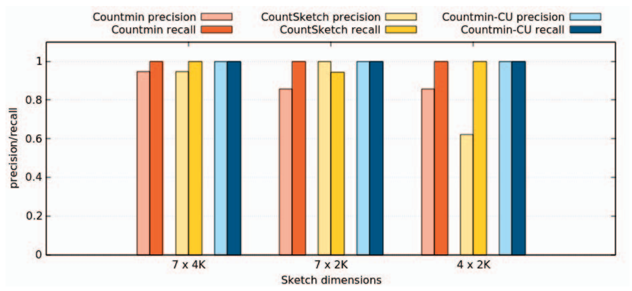


Fig. 7. Sketch performance with the CAIDA database.

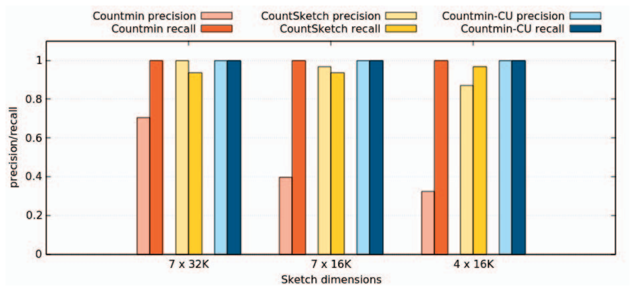


Fig. 8. Sketch performance with the mESC database.

Fig. 7 shows precision and recall obtained with our software framework for the three sketches with the CAIDA database using a counter matrix of size $7 \times 4,096$, $7 \times 2,048$, and $4 \times 2,048$. The graph shows that, for large memory sizes, all sketches show good performance, partly because there are comparatively few low-frequency elements. The precision of CM degrades gracefully as its storage memory size decreases. The recall of CS is relatively robust to memory size, but its precision degrades to around 0.6 with the smallest memory. CM-CU shows perfect precision and recall for all memory sizes. Below the memory sizes shown in the graph, the performance of CM-CU worsens considerably.

Fig. 8 shows the results for the mESC database with a counter matrix of $7 \times 32,768$, $7 \times 16,384$, and $4 \times 16,384$. Given frequency distribution of the mESC data, CM shows very poor precision for all memory sizes, because the error induced by sketch collisions with low-frequency elements causes the algorithm to overestimate the frequency of these elements, thus increasing the number of false positives. The performance of CS is more robust, but its precision still degrades by more than 15% with a 4×16K-counter matrix. Again, precision and recall of CM-CU is 1 in all cases.

TABLE II. SOFTWARE VERSUS PIPELINED CM-CU

Counter matrix size	7×4K	4×4K	7×2K	4×2K
Software CM-CU E_{HH}	0.08%	0.11%	0.27%	0.34%
Software CM-CU precision	1	1	1	1
Software CM-CU recall	1	1	1	1
Pipelined CM-CU E_{HH}	0.11%	0.12%	0.28%	0.35%
Pipelined CM-CU precision	1	1	1	1
Pipelined CM-CU recall	1	1	1	1

In Section IV-B, we showed that the data hazard introduced by the two-cycle latency of the sketch update pipeline in Contmin-CU could produce errors in the frequency estimates. We argued that the probability of this hazard is very low and chose not to include a hazard detection and stalling/forwarding mechanism that could potentially degrade the performance of the hardware. In order to test this assertion, we define the relative estimation error produced by the sketch as:

$$E(x_i, \hat{x_i}) = \frac{|x_i - \hat{x_i}|}{x_i} \times 100 \tag{6}$$

where x_i is the true frequency of the element, and $\hat{x_i}$ is the frequency estimation produced by the sketch. To summarize the relative error in the dataset, we define the mean estimation error as the geometric mean of the estimation error over the entire set of heavy hitters:

$$E_{\text{HH}} = \sqrt[n]{\prod_{i=1}^{n} E(x_i, \hat{x_i})}, \forall i \in \text{HH} \tag{7}$$

where HH is the set of element of the dataset classified as heavy hitters using the true frequency of the elements. We restrict the metric only to the heavy hitters to measure the distance to validity, according to the criterion defined in [20].

Table II shows the mean estimation error for both the software version of CM-CU and for the pipelined implementation of the algorithm without hazard detection, using the CAIDA database. The table shows that, although collisions do indeed occur, their impact on the frequency estimate of the heavy hitters is smaller than 0.03% even for the smallest memory sizes. This error is insufficient to cause changes in the selection of heavy hitters and, as a result, precision and recall for both versions of the algorithm is 1.

TABLE III. KINTEX 7 XC7K325T FPGA RESOURCE UTILIZATION

	Full sketch		Hashes		Matrix/Updates		DDR control	
Logic	Used	%	Used	%	Used	%	Used	%
LUTs	11,039	5.4	390	0.19	236	0.1	10,413	5.1
Regs	8,859	2.2	945	0.23	250	0.06	7,664	1.9
BRAM	38.5	8.6	—-	—-	20	4.5	18.5	4.2
BUFG	5	15.6	—-	—-	—-	—-	—-	—-
MMCM	2	20	—-	—-	—-	—-	—-	—-

TABLE IV. RESOURCE UTILIZATION OF DIFFERENT SKETCHES

	mESC database			CAIDA database		
	CM	CS	CM-CU	CM	CS	CM-CU
Sketch size	7x32k	7x16k	4x16k	7x32k	7x16k	4x16k
% LUTs	0.61	0.62	0.31	0.41	0.49	0.22
% Regs	0.55	0.63	0.29	0.44	0.51	0.23
% BRAM	15.7	7.86	4.49	2.35	1.57	0.89
Precision	0.70	0.96	1	0.94	1	1
Recall	1	0.93	1	1	0.94	1

C. *Hardware performance*

We built a hardware prototype of the sketch architecture described in Section IV. We designed the circuit using a register-transfer level (RTL) description in the SystemVerilog hardware description language. We synthesized and mapped the design onto a Xilinx Kintex-7 XC7K325T FPGA with DDR3 external memory, using the Xilinx Vivado Suite.

The prototype implements the CM-CU sketch using four rows of 16,384 11-bit counters, which is sufficient to operate with both our testing databases, as shown in Section V-B. The latency of the circuit due to the pipelined hash functions and sketch updates is 25 clock cycles. Table III shows the logic resource utilization of our implementation. The table shows the resource usage of the complete sketch, as well as a breakdown divided into hash functions, matrix storage and counter update circuits, and the memory controller. As the table shows, the design features very low resource utilization, with most of the slice logic being consumed by the memory controller and buffer. The exception is the counter matrix, which naturally uses more blockRAM resources: With $4 \times 16K$ counters, the sketch requires four blockRAM modules to store the data. The design uses less than 6% of the logic and memory resources on the chip, and no DSP slices. Because the utilization of global resources such as clock managers and buffers does not scale with sketch size, we could use the available logic to increase the number of counters or hashes, replicate the sketch to handle multiple data streams simultaneously, or perform further processing on the heavy hitters on chip.

Table IV compares the resource utilization of hardware implementations of the CM [1], CS [14], and CM-CU sketches on the Kintex-7 FPGA. The size of the sketches were chosen to achieve comparable performance for each data set, as shown in Figs. 7 and 8. Resource utilization is mostly determined by sketch size independently of the update algorithm. Because of its lower memory requirements, CM-CU achieves better performance with lower resource utilization than CM and CS.

Our prototype runs with a 300MHz clock, and the critical path is in the memory controller. The sketch core can run at nearly 400MHz, thanks to the pipelined implementation of the hash functions and sketch-update logic, and to its low blockRAM utilization. Section V-B shows that CM and CS require more memory to achieve comparable performance,

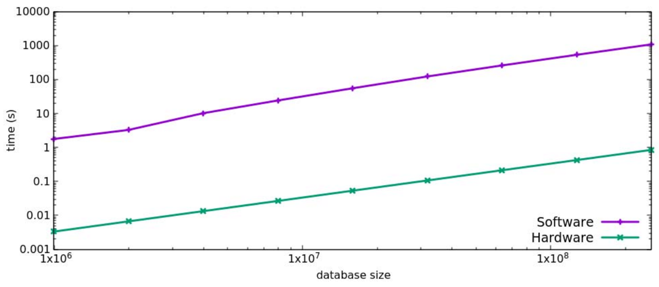


Fig. 9. Execution time with data from the mESC database (log-log scale)

therefore this is a clear advantage of using the Countimin-CU algorithm. Because the sketch accepts a new input every clock cycle, it can process the CAIDA data at a peak performance of 300Mpackets/s, which enables it to perform packet inspection in real time at a line rate of more than 100Gbps.

The power dissipation of the FPGA reaches 2.041W, of which 91% is dynamic power. The DDR3 controller consumes 1.76W of dynamic power and the sketch core consumes 97mW (44mW in the hash functions, 36mW in the blockRAMs, and 18mW in the update logic).

In order to test the performance of the prototype as a hardware accelerator for Chip-seq data analysis, we performed heavy-hitter detection on the mESC database, using k-mers of length 16, which are represented by the sketch using 32 bits, as discussed in Section V-A. We created datasets of different sizes by subsampling and replicating the original database. We measured the time required to process the complete stream on the FPGA after the data was uploaded to its external memory. We also measured the execution time of the optimized software version of the algorithm on a desktop computer with an Intel Core i5-8600K processor running at 4.3GHz and 32GB of DDR4 2133MHz memory, running a 4.13.43 Linux kernel. We measured the execution time on the computer after the dataset has been uploaded to system memory as well.

Fig. 9 shows the execution time versus database size in a log-log scale. As expected, the execution time of the hardware prototype scales linearly with the number of elements in the dataset. The execution time on the computer is also linear once the dataset becomes large enough to make the benefits of the processor cache negligible. With the original database, the FPGA detects the heavy hitters in 13.27ms, while the computer requires 10.2s to process the same data, which corresponds to a speedup of 768. With the subsampled and scaled-up versions of the database, the speedup varies from 535 to 1,274.

D. *Discussion*

The published literature for hardware acceleration of heavy-hitter detection has focused on network traffic analysis. An accelerator [14] based on a Xilinx Virtex Ultrascale XCVU095 FPGA reaches 100Gbps line rate using a $8 \times 1M$ CS structure and a priority queue to detect the top-N heavy hitters. They report an 1.29% average error in their frequency estimates with 300,000-packet traces in the CAIDA database. In comparison, our solution achieves 0.12% error for the same trace sizes, running at the same or higher data throughput, and with a much smaller ($4 \times 16K$) sketch and a smaller FPGA.

Another application in network traffic anomaly detection shows a hardware implementation of counting sketches [1] for heavy hitter detection and change. They achieve a throughput of 150Gbps 10×32K CM and K-ary sketches for heavy hitter detection using a Xilinx Virtex Ultrascale XCVU440 FPGA while guaranteeing a probability of false positives lower than 0.001. In our experiments, the CM-CU sketch produces no false positives with a 4×16K sketch, and can achieve a throughput of up to 128Gbps when the sketch core runs at 400MHz. Mapping our design to a high-end FPGA such as the Virtex Ultrascale line would likely yield better performance results. One notable feature of the architecture presented in [1] is that, in order to achieve high throughput, the authors use a deep pipeline to access the sketch matrix when its size is equal to or larger than 32K rows. The depth of the pipeline requires the use of a complex hazard detection and data forwarding unit to guarantee consistency in the counter updates. The CM-CU algorithm can operate with a smaller memory and only uses a 2-stage pipeline to perform the counter updates, which eliminates the need for the forwarding unit.

VI. Conclusions

We have described a special-purpose architecture for online heavy hitter detection on large data streams. The architecture estimates element frequencies in the stream using a CM-CU sketch. The frequency estimates are compared to a user-supplied threshold to select the heavy hitters, although other data structures, such as a fixed-size heap or priority queue can be used to select the top-N hitters. The CM-CU algorithm allows us to select heavy hitters with guaranteed perfect recall, and higher precision and lower memory requirements than other algorithms for sublinear-space approximate counting.

Our implementation on a Xilinx Kintex-7 XC7K325T FPGA accessing data from external DDR3 memory runs at with a system clock of 300MHz. It can be used for network traffic analysis at line rates exceeding 100Gbps. We also tested the accelerator to detect frequent DNA sequences on Chip-seq data, achieving a speedup between 535 and 1,274 when compared to a modern desktop computer. Our architecture uses less than 6% of the resources available on the chip, leaving space to host multiple parallel sketches, which can be useful when looking for motifs in DNA sequences, or to add more hardware units to perform further processing on the heavy hitters, such as a fixed-size heap or priority queue, or additional sketches for heavy change detection on network flow data.

We are currently extending the architecture to discover motifs in Chip-seq DNA data, testing other sketches, and adding heap structures to store and process the heavy hitters.

Acknowledgments

This work was supported by CONICYT, Chile, through FONDECYT grant 1180995 and Basal Funds FB0001. A. Saavedra acknowledges the support of CONICYT-PFCHA Magíster Nacional 2015 scholarship 22170799.

References

[1] D. Tong and V. K. Prasanna, "Sketch acceleration on fpga and its applications in network anomaly detection," *IEEE Transactions on Parallel & Distributed Systems*, vol. 29, no. 4, pp. 929–942, April 2018. [Online]. Available: doi.ieeecomputersociety.org/10.1109/TPDS.2017.2766633

[2] Q. Zhang, J. Pell, R. Canino-Koning, A. C. Howe, and C. T. Brown, "These are not the k-mers you are looking for: Efficient online k-mer counting using a probabilistic data structure," *PLOS ONE*, vol. 9, no. 7, pp. 1–13, 07 2014. [Online]. Available: https://doi.org/10.1371/journal.pone.0101271

[3] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 123–137.

[4] Y. Li, R. Miao, C. Kim, and M. Yu, "Flowradar: A better netflow for data centers." in *NSDI*, 2016, pp. 311–324.

[5] P. Melsted and J. K. Pritchard, "Efficient counting of k-mers in dna sequences using a bloom filter," *BMC bioinformatics*, vol. 12, no. 1, p. 333, 2011.

[6] Q. Yu, H. Huo, X. Chen, H. Guo, J. S. Vitter, and J. Huan, "An efficient algorithm for discovering motifs in large dna data sets," *IEEE transactions on nanobioscience*, vol. 14, no. 5, pp. 535–544, 2015.

[7] P. Audano and F. Vannberg, "Kanalyze: a fast versatile pipelined k-mer toolkit," *Bioinformatics*, vol. 30, no. 14, pp. 2070–2072, 2014.

[8] R. Berinde, P. Indyk, G. Cormode, and M. J. Strauss, "Space-optimal heavy hitters with strong error bounds," *ACM Transactions on Database Systems (TODS)*, vol. 35, no. 4, p. 26, 2010.

[9] C. Estan and G. Varghese, "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice," *ACM Transactions on Computer Systems (TOCS)*, vol. 21, no. 3, pp. 270–313, 2003.

[10] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," *Theoretical Computer Science*, vol. 312, no. 1, pp. 3 – 15, 2004, Automata, Languages and Programming. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304397503004006

[11] M. Muthukrishnan and G. Cormode, "Approximating data with the count-min sketch," *IEEE Software*, vol. 29, pp. 64–69, 2011.

[12] D. Tong and V. Prasanna, "High throughput sketch based online heavy hitter detection on fpga," *SIGARCH Comput. Archit. News*, vol. 43, no. 4, pp. 70–75, Apr. 2016. [Online]. Available: http://doi.acm.org/10.1145/2927964.2927977

[13] D. Nguyen, G. Memik, S. O. Memik, and A. Choudhary, "Real-time feature extraction for high speed networks," in *International Conference on Field Programmable Logic and Applications, 2005.*, Aug 2005, pp. 438–443.

[14] J. F. Zazo, S. Lopez-Buedo, M. Ruiz, and G. Sutter, "A single-fpga architecture for detecting heavy hitters in 100 gbit/s ethernet links," in *2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Dec 2017, pp. 1–6.

[15] M. V. Ramakrishna, E. Fu, and E. Bahcekapili, "Efficient hardware hashing functions for high performance computers," *IEEE Transactions on Computers*, vol. 46, no. 12, pp. 1378–1381, Dec 1997.

[16] J. Carter and M. N. Wegman, "Universal classes of hash functions," *Journal of Computer and System Sciences*, vol. 18, no. 2, pp. 143 – 154, 1979. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0022000079900448

[17] Q. Yu, H. Huo*, X. Chen, H. Guo, J. S. Vitter, and J. Huan, "An efficient algorithm for discovering motifs in large dna data sets," *IEEE Transactions on NanoBioscience*, vol. 14, no. 5, pp. 535–544, July 2015.

[18] C. for Applied Internet Data Analysis. (2016) Caida data. [Online]. Available: www.caida.org

[19] X. Chen, H. Xu, P. Yuan, F. Fang, M. Huss, V. B. Vega, E. Wong, Y. L. Orlov, W. Zhang, J. Jiang, Y.-H. Loh, H. C. Yeo, Z. X. Yeo, V. Narang, K. R. Govindarajan, B. Leong, A. Shahab, Y. Ruan, G. Bourque, W.-K. Sung, N. D. Clarke, C.-L. Wei, and H.-H. Ng, "Integration of external signaling pathways with the core transcriptional network in embryonic stem cells," *Cell*, vol. 133, no. 6, pp. 1106–1117, June 2008. [Online]. Available: http://dx.doi.org/10.1016/j.cell.2008.04.043

[20] G. T. Minton and E. Price, "Improved concentration bounds for count-sketch," *CoRR*, vol. abs/1207.5200, 2012. [Online]. Available: http://arxiv.org/abs/1207.5200