Memory Aware Packet Matching Architecture for High-Speed Networks

Michal Kekely
FIT BUT
Božetěchova 2, 612 66 Brno
Czech Republic
ikekelym@fit.vutbr.cz

Lukáš Kekely
CESNET, a. l. e.
Zikova 4, 160 00 Prague 6
Czech Republic
kekely@cesnet.cz

Jan Kořenek
IT4Innovations Centre of Excellence, FIT BUT
Božetěchova 2, 612 66 Brno
Czech Republic
korenek@fit.vutbr.cz

***Abstract*—Packet classification is a crucial operation for many different networking tasks ranging from switching or routing to monitoring and security devices like firewall or IDS. Generally, accelerated architectures implementing packet classification must be used to satisfy ever-growing demands of current high-speed networks. Furthermore, to keep up with the rising network throughputs, the accelerated architectures for FPGAs must be able to classify more than one packet in each clock cycle. This can be mainly achieved by utilization of multiple processing pipelines in parallel, what brings replication of FPGA logic and more importantly scarce on-chip memory resources.**

Therefore in this paper, we propose a novel parallel hardware architecture for hash-based exact match classification of multiple packets per clock cycle with reduced memory replication requirements. The basic idea is to leverage the fact that modern FPGAs offer hundreds of BlockRAM tiles that can be accessed (addressed) independently to maintain high throughput of matching even without fully replicated memory architecture. Our results show that the proposed approach can use memory very efficiently and scales exceptionally well with increased record capacities. For example, the designed architecture is able to achieve throughput of more than 2 Tbps (over 3 000 Mpps) with an effective capacity of more than 40 000 IPv4 flow records for the cost of only 366 BlockRAM tiles and around 57 000 LUTs.

I. Introduction

With the increasing capacity of network links, all network devices and systems need to speed up their packet processing. Current processors are not able to cope with network traffic even on 100 Gbps links. In order to achieve wire-speed processing with a throughput of 100 Gbps and more, network systems have to utilize FPGA or ASIC technology. The FPGA acceleration provides high performance and is highly configurable (flexible) as well. The flexibility is essential for any practical network system because traffic processing is changing with the introduction of every new protocol, application or service. Therefore, 40 Gbps network interface cards with FPGAs started to be deployed to data centers as hardware platforms for the acceleration [1] and will be probably more and more frequently used in the future.

Network traffic processing architecture can be easily described in the P4 high-level language [2] and then automatically mapped directly to an FPGA hardware accelerator [3], [4]. The P4 language has been designed at Stanford University in order to enable protocol, vendor and target independent definitions of packet processing. An integral part of the P4 language is the utilization of match/action tables as a basis to control processing of each input packet.

The match/action tables perform various forms of packet classification. During the classification, packets are matched against a set of rules, which are usually defined by values, ranges or prefixes of a few selected packet header fields. Generally, the classification is a mathematical problem of a multidimensional range search. Due to the ruleset size and complexity of rules, it is very difficult to perform matching at sufficient rate for wire-speed processing. Therefore, many hardware architectures have been designed to accelerate packet classification [5], [6], [7], [8], [9], [10], [11].

For 100 Gb network links, wire-speed throughput can be achieved only if a new packet is processed every 6.7 ns, which is only one clock cycle for 150 MHz clock. It means that multiple packets have to be processed within each clock cycle to achieve wire-speed 400 Gbps or 1 Tbps packet processing in FPGAs. Usually, the processing speed is increased by utilization of multiple parallel pipelines [12], [13], which require multi-port memories or memory replication. Unfortunately, both approaches significantly reduce throughput scalability at 400 Gbps or 1 Tbps speeds.

Therefore, we focus on the design of a new hardware acceleration technique for packet classification with efficient utilization of memory resources to achieve high-speed packet processing. We introduce novel hardware architecture that is able to scale the throughput of P4 match/action tables to more than 2 Tbps (over 3 000 Mpps) on current FPGAs while memory replication is significantly reduced compared to other approaches. The proposed concept is compared with simple pipeline/memory replication scheme and several possible optimizations are introduced.

II. Related Work

Currently, there are many different approaches to packet classification. Some of them focus on being as general as possible, supporting packet classification in multiple different dimensions and different types of match strength, such as range lookups, ternary matching or longest prefix match (LPM). However, the only way how to scale most of those approaches for higher throughputs is to utilize multiple copies of the same architecture operating in parallel.

Packet classification based on bit-parallelism (or bit vectors, BV), proposed by Lakshman et al. [14], is a practical implementation that leverages the fact that rule updates are infrequent compared to search operations. The algorithm works in two stages. In the first stage, parallel searches are carried out within each of the dimensions, resulting in bit vectors. Each bit of these vectors corresponds to one record in classification ruleset, therefore their width is given by the number of rules used. A bit is set to one if a corresponding rule is matched in given dimension and is reset to zero otherwise. After the first stage, every bit vector represents the set of all the rules matched in one dimension. Then the second stage has to find an intersection of the sets matched within single dimensions. Since these sets are represented as bit vectors finding the intersection is reduced to bitwise AND operation among the bit vectors. The main problem with this approach is the width of bit vectors which increases with the number of rules. Song et al. [15] presented architecture that combines bit vector approach with TCAMs. The architecture uses TCAMs for lookups within dimensions that require exact or prefix matches and tree-bitmap implementation of the BV algorithm for source and destination port lookups. This architecture is optimized for classification based on network flow 5-tuples (source IP address, destination IP address, source port, destination port and L4 protocol), therefore it is not very flexible and was not shown to have the ability to scale to support different header fields.

Several of different approaches to supporting multiple dimensions are described in [16]. A grid of Tries extends standard Trie structure to two dimensions however, it is not easily extendable to more than two. General solution using cross-products is more promising, but with no further optimizations uses up way too much memory and resulting cross-products are quite big. Other trie-based algorithms scale poorly with increasing number of dimensions. Additionally, these algorithms need great amounts of memory and cannot be easily scaled to higher throughputs.

Another group of approaches to classification tries to utilize architectures based on the construction of decision trees. Many of these algorithms are not designed with FPGA implementation in mind, however, some of them can be bent to be efficiently mapped into FPGA structure. HiCuts [6] and HyperCuts [7] are examples of such algorithms. The main idea is to progressively cut the whole searched space represented by classification dimensions into small enough parts (usually representing 1 or several rules). Different heuristics can be used to decide how to cut the space. But, resulting trees tend to have many nodes. Additionally adding or removing rules leads to the need of rebuilding the whole tree.

Prasanna et al. [17] pushed the idea of constructing decision trees even further. They have observed that HyperCuts and similar algorithms do not efficiently deal with rules that have too much overlap with each other. In such cases, many rules need to be duplicated and the resulting tree (and required memory) can explode exponentially with the number of dimensions. To combat this, a decision forest is introduced. Ruleset is split into subsets and smaller decision trees are built for each subset. Rules within each subset are chosen so that they have as little overlap as possible and that they specify nearly the same dimensions. Additionally, two other techniques are used to optimize HyperCuts algorithm. Rule overlap reduction stores rules that should be replicated in a list in each internal node instead of actually replicating it into all the child nodes. Precise range cutting is used to determine cutting points which will result in the least number of rule duplications instead of deciding number of cuts for a field.

Taylor et al. [5] introduced Distributed Crossproducting of Field Labels (DCFL). This algorithm decomposes classification into single dimensions and can be easily parallelized. Moreover, it uses Bloom Filters [9] and labeling technique to lower memory and logic requirements. The architecture was shown to be scalable even to higher throughputs [18], but only by using multiple copies of the memories. Because of this key features, the architecture can be duplicated to increase throughput while still maintaining reasonable usage of on-chip memories and logic. This idea was pushed further to build scalable architecture through memory duplication in [18].

In many cases, exact match packet classification is sufficient. This is prevalent mainly when IP flows are concerned. Effective approaches to exact match packet classification are usually based on hash tables. A sophisticated way of implementing hash tables is cuckoo hashing principle [19]. The main idea of cuckoo hashing is to increase the efficiency of memory utilization in the hash table by multiple parallel hash functions/tables. Each table uses one of the different hash functions for indexing. This means that if a new element cannot be inserted into the first table because of a conflict with an already existing item, it can still be inserted into one of the other tables through a different hash function. Even when the element cannot be inserted into any of the tables it can still be inserted by force, pushing out one of the previous occupants. The previous occupant can then be reinserted into the tables in the same manner. Using more tables and reinsertions allows the cuckoo hashing to keep the high lookup speed while decreasing the number of unresolvable conflicts and therefore increasing the effective capacity.

The cuckoo hashing approach is well suited for hardware because each hash table can work in parallel [20], [21]. These published implementations offer throughputs up to around only 100 Gbps, while in this paper we aim at achieving over 1 Tbps. Cuckoo hashing based packet classification is also effectively used to monitor or analyze network traffic in the idea of Software Defined Monitoring (SDM) [22]. Here, an external memory is utilized and achieved throughput is again shown to be sufficient only for up to 100 Gbps.

III. Architecture

Our main goal is to design an architecture for exact match packet classification that can accommodate high throughputs of multiple terabits per second. One way to achieve this would be to increase the clock frequency of basic cuckoo hashing architecture. This is possible to do only until a certain

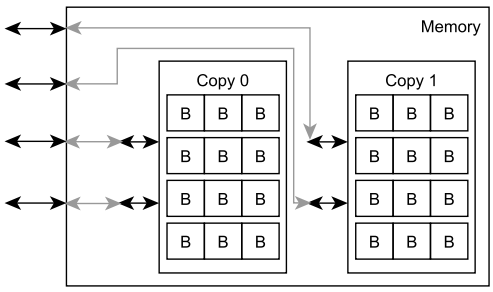

Fig. 1. The memory architecture of simple replication approach.

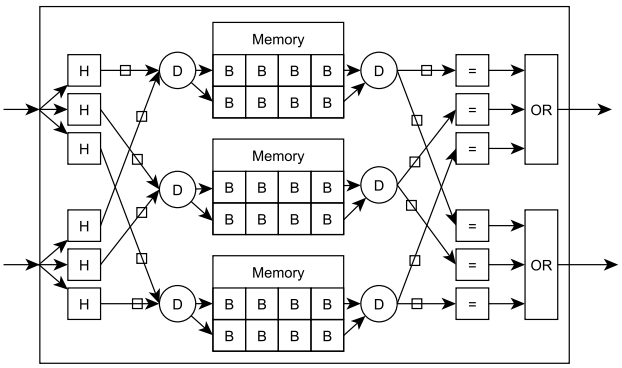

Fig. 2. The top-level architecture of the proposed optimization.

point, after which the frequency cannot be further increased due to the limitations of used FPGA technology. The other way to increased throughput is through a design of the new architecture of cuckoo hashing that can carry out more than one rule lookup per each clock cycle. This translates to more than one memory access per clock cycle to each utilized hash table. Current Xilinx FPGAs use BlockRAM tiles as the main type of on-chip memories. These have 2 independent ports, therefore we can easily perform 2 memory accesses per clock cycle with no additional cost.

In order to enable more than 2 accesses per clock cycle, we can simply replicate the memories. This is illustrated for 4 accesses in the figure 1. Two of the accesses are mapped to one copy of the memory and two are mapped to the other copy. This approach is not particularly efficient because we need to double the memory in order to achieve doubled throughput. However, we can leverage the internal structure of FPGAs and their memory tiles. A single copy of a larger memory is internally usually composed of more than one BlockRAM tile (B blocks). Each BlockRAM on current Xilinx chips [23] can be used as 36 b wide dual port memory with 1 024 entries. Larger memories are constructed utilizing multiple BlockRAMs organized into several rows and columns. For example, figure 1 corresponds to data width of up to 108 b and 4 048 entries.

A. Proposed Approach

If we already have more than one row of BlockRAMs in each table we should be able to do more than just two memory accesses per clock cycle. We can, in fact, ideally do two accesses per cycle independently to each of the individual rows. This fact can be leveraged to optimize the previously mentioned simple replication approach. We propose an FPGA architecture of cuckoo hashing shown in figure 2. The proposed approach is also applicable to any other kinds of hash tables, but we choose cuckoo hashing as it is the most effective existing hashing scheme.

The figure shows the architecture able to carry out up to 2 parallel lookups per cycle with cuckoo hashing using 3 different hash functions/tables. The memory blocks used here are similar to the blocks from figure 1 – meaning that they internally consist of multiple independent rows of BlockRAMs. Hash functions are computed for each lookup key (H blocks) and are connected to a distribution logic (D blocks). There is one distributor block for each hash function/table of the cuckoo hashing. The distributor consists primarily of logic that maps the requested memory accesses into corresponding table rows given by a few most significant bits of their hash values and distributes them onto available BlockRAM ports for each of these rows. On the other side, it correctly forwards read data from each memory row and port to the corresponding comparison logic (= and OR blocks). The basic idea is to replicate memory fewer times than in the case of a simple approach (fewer replicas than required parallel packet lookups) as we can perform multiple accesses per clock cycle simply by hash functions that are pointing into different table rows. Additionally, memory can also be replicated here to enable more than two parallel access ports for each row.

So, the distributor blocks determine which row of the BlockRAMs is accessed by each lookup and sets the control logic in a way to carry out all the lookups that are not in conflict with each other. Conflicts, in this case, mean that there are more lookups wanting to access the same row of one table than there are available access ports in this row. Note that since the memories can still be replicated the number of available access ports might be higher than two. All the lookups that could not be carried out in the first cycle will be carried out in consecutive cycles until all of the requested lookups are finished. This means that the lookup of all of the inputs might take more than one cycle. However, the basic idea is that the relative number of occurring conflicts (or rather number of additional cycles needed) is pretty low for higher numbers of memory rows, thus reducing throughput only slightly. Compared to that, the saved memory resources thanks to no or weaker memory replication are considerable.

For example, consider a case where four lookups each clock cycle are needed and there are four rows of BlockRAMs with only two ports each (meaning no memory replication). There are no conflicts unless at least three of the four lookups need to access the same row. In case of the conflict, two of the conflicting accesses can still be carried out together with all others that are not in conflict. The last one or two

accesses from the conflicting group has to be carried out in the next cycle. Even if there is a conflict every time, we still achieve the same throughput as the simple architecture with the same memory requirements (replication factor). In the example without replication, we would do four lookups in two clock cycles which is the same as the simple approach with two lookups each cycle. This shows that at worst the proposed solution is on par with the simple solution in terms of both memory and throughput. However, the key idea is that the conflicts do not occur each time and are actually pretty infrequent (20 % conflict chance in this example), therefore the effective achieved throughput is considerably better.

It is also important to note that we can easily achieve independence in the conflict handling for each parallel hash table used in the cuckoo hashing. The distributor corresponding to a single hash table does not need to wait until all the other distributors carried out all their lookups. Instead, there are small input and output buffers that are used to synchronize the results (denoted by squares on corresponding connections in figure 2). This makes the architecture a lot more efficient as the throughput is not governed by the probability of no conflicts in all of the tables together but rather by the probability that there are no conflicts in every single table independently. This independent probability is a lot lower especially when a higher number of parallel hash tables are used.

Of course, the described buffers consume some additional FPGA resources. Also, the distributors themselves introduce some logic overhead compared to simple replication approach. In the simple approach, there is a dedicated port for each parallel lookup, therefore hash functions (inputs) and comparison logic (outputs) can be directly connected to appropriate memories without the distributors. The core of each distributor is a planner, that can evaluate and resolve access conflicts – basically a group of encoders and decoders to select a valid access plan for each cycle. The planner controls two columns of multiplexers: the first to route planed access requests to correct memory rows/ports and the second to pair read data with their corresponding requests. Additional registers are used to thoroughly pipeline the distributors for better frequency and to correctly synchronize all operations together. The total FPGA logic overhead of the distributors and buffers is expected to be manageable compared to complex hashing blocks which are usually considerably large and contain critical paths.

The described architecture can be optimized for better throughput even further as during conflicts the available access ports of memories are currently not fully utilized in the added clock cycles. For example, if only one lookup cannot be carried out in the first cycle it has to be carried out in the second (additional) one. Reserving one full clock cycle just for one extra lookup is inefficient. A more reasonable approach would be to combine the extra lookup cycles with some of the lookups needed for the next inputs. While this cycle sharing increases the throughput by a small percentage it requires a lot more complex distributor and buffer architectures. Because of this the rest of the paper deals with the architecture without such optimization.

B. Analysis of Conflicts

It is possible to mathematically analyze the probability of conflict occurrence and derive the achievable throughput of the proposed architecture with given parameters. There are 3 main parameters of the architecture when it comes to the probability of conflicts: the number of rows of BlockRAMs in each table r, the number of parallel lookups per clock cycle l corresponding to the number of inputs, and the number of available access ports for each table row p.

The probability that a single lookup needs to access one specific selected row and the probability that it needs to access any other row are complementary:

$$p_s(r) = \frac{1}{r} \tag{1}$$

$$p_{ns}(r) = \frac{r-1}{r} \tag{2}$$

First of all, for any given n the probability that exactly n lookups out of total l in one cycle need to access one selected row out of r rows can be computed as a product of: the probability that selected n lookups access selected row, the probability that all the other $l-n$ lookups do not access this row, and the number of combinations by which it is possible to position those n lookups into all l. The appropriate equation:

$$\begin{aligned} p_s(n,l,r) &= (p_s(r))^n * (p_{ns}(r))^{l-n} * \binom{l}{n} \\ &= \left(\frac{1}{r}\right)^n * \left(\frac{r-1}{r}\right)^{l-n} * \binom{l}{n} \end{aligned} \tag{3}$$

To get the probability that any of the rows will have exactly n lookups mapped onto it we simply multiply the previous probability from equation 3 by the number of rows:

$$\begin{aligned} p_a(n,l,r) &= p_s(n,l,r) * r \\ &= \left(\frac{1}{r}\right)^{n-1} * \left(\frac{r-1}{r}\right)^{l-n} * \binom{l}{n} \end{aligned} \tag{4}$$

Now to approximate the probability that more than n lookups out of all l in one cycle need to access the same row out of r we can simply sum the probabilities from equation 4 for all values higher than given n:

$$\begin{aligned} p_{c,a}(n,l,r) &= \sum_{i=n+1}^{l} p_a(i,l,r) \\ &= \sum_{i=n+1}^{l} \left(\frac{1}{r}\right)^{i-1} * \left(\frac{r-1}{r}\right)^{l-i} * \binom{l}{i} \end{aligned} \tag{5}$$

This sum does not account for the fact that solution spaces described by some of the summed probabilities have non-empty intersections with one another (some conflict variants are counted multiple times). To counter this fact we would have to compute probabilities that exactly n lookups will be mapped onto the same row while there is no other row with n or more lookups mapped onto it. This leads to exponentially more complex nested sums. However, the approximate results achieved by the equation 5 are always higher than the actual

results, which means they would actually give us more pessimistic results for the throughput. Additionally this approximation is very precise for results under configurations that are the most interesting for us. For example, it is absolutely precise if p is higher or equal to $l/2$, since in this case, it is impossible for two different rows to have more than p accesses mapped at the same time.

The equation 5 essentially approximates the probability that there will be a conflict for architecture with l lookups, r rows of BlockRAMs and p=n ports for each row. However, not all conflicts are equal when it comes to their effect on the achieved throughput. For example, if $p = 2$ and 6 lookups need to access the same row it takes 3 cycles to carry out all of all them, while when 4 lookups need to access the same row only 2 cycles are needed. To extend our equations and reflect this we use a weighted sum:

$$c_{w,c}(n,l,r) = \sum_{i=n+1}^{l} w(i,n) * p_a(i,l,r) \tag{6}$$

The weight w here represents the number of cycles needed to resolve the conflict in each case:

$$w(i,n) = \left\lceil \frac{i}{n} \right\rceil \tag{7}$$

Finally we can do one last thing to get how many times more cycles (on average) are needed compared to the case without any conflicts. The equation 6 sums only weighted probabilities of conflicts. We need to add also the probability that there will be no conflict. Weight corresponding to no conflict is obviously 1 since even when there is no conflict we still need one clock cycle to carry out all the lookups. So the coefficient that gives us the relation between needed cycles (achieved throughputs) is computed as follows:

$$c(n,l,r) = c_{w,c}(n,l,r) + (1 - p_{c,a}(n,l,r)) \tag{8}$$

In conclusion, the proposed optimized architecture with l lookups, r BlockRAM rows, and p ports can achieve throughput equivalent to an average of m lookups per cycle, where:

$$m = \frac{l}{c(p,l,r)} \tag{9}$$

Thanks to the previously mentioned buffers there is no need to include number of hash functions (parallel hash tables) into our computations. Logic and memories corresponding to each hash operate independently of one another and their results are only synchronized afterward via buffers. This means that if there is a collision in memory tied to one hash another hash with no collision does not have to wait.

IV. Results

The results in this section are obtained through the previously mentioned mathematical analysis and are confirmed through experiments with implemented architecture. Measurements are based on design synthesis for the Xilinx UltraScale+ XCVU9P FPGA [23] using Vivado 2017.4 tool. The architecture is able to achieve working frequency (F_{max}) of more than 400 MHz for every evaluated configuration. Therefore, the following throughput results are all shown for 400 MHz clock frequency. All the cases used 104 b wide key that is sufficient for the classification of standard IPv4 flows (5-tuple) and 32 b wide arbitrary data (action). There are 3 main parameters that are worth exploring in the results – resource requirements (BRAMs, LUTs), achievable throughput (lookups per cycle, Mpps, Gbps), and effective rule capacity.

Hash functions	Rows of BRAMs	Total capacity	Effective capacity
3	1	3 072	2 765
3	2	6 144	5 530
3	4	12 288	11 059
3	8	24 576	22 118
3	16	49 152	44 236
4	1	4 096	3 891
4	2	8 192	7 782
4	4	16 384	15 565
4	8	32 768	31 130
4	16	65 536	62 260

TABLE I
CAPACITIES OF CUCKOO HASHING FOR DIFFERENT PARAMETERS.

Table I shows different capacities of cuckoo hashing architecture based on the number of hash functions and the number of BlockRAM rows for each table (each row has 1 024 items). Total (theoretical) capacity and achievable effective (mean) capacity are shown. For three functions the efficiency of capacity utilization is around 90 %, for four it is around 95 %. This is consistent with similar measurements in [21]. Table I is primarily used to illustrate cuckoo hashing capacities that are considered in the evaluation.

Figure 3 captures the relation between throughput and memory requirements of architectures with three hash functions in different configurations. Lines in the graph represent throughput and memory requirements of simple memory replication approach for a different number of BlockRAM rows used. Again, the number of rows is directly tied to the capacity of the architecture as shown in table I. These results form a baseline for evaluation of the designed optimization.

Each point in the graph shows results for a different configuration of the proposed memory optimized approach. The color of a point represents the number of BlockRAM rows used (the capacity of the architecture) and its shape represents how many lookups (number of inputs l) the architecture supports. Our approach is clearly better in terms of used memory for each given throughput achieved as all points are below lines of appropriate color. Obviously, when there is only one row of BlockRAMs (black line) there is no possibility to employ our optimization and gain something. However, even when there are only 2 rows of BlockRAMs (light blue) we can already achieve better results. For example, using an architecture with 10 lookups (full circles) we can achieve 48.5 % increase in throughput without any memory duplication.

The results tend to get even better when using more rows of BlockRAMs. This is expected behavior since more rows mean

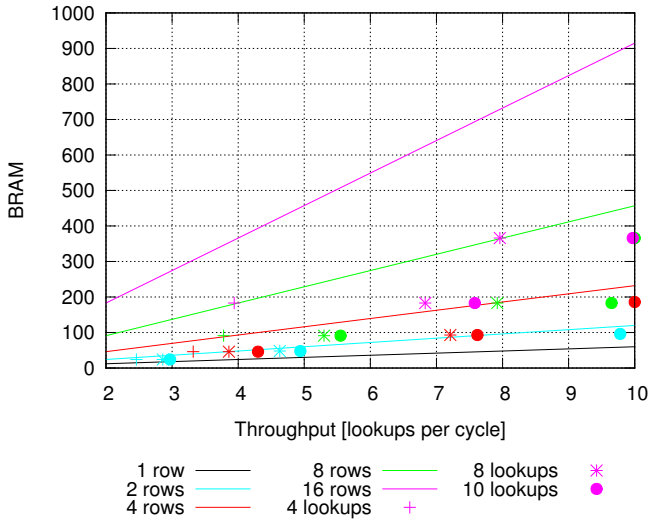

Fig. 3. The relation between memory and throughput for 3 hash functions.

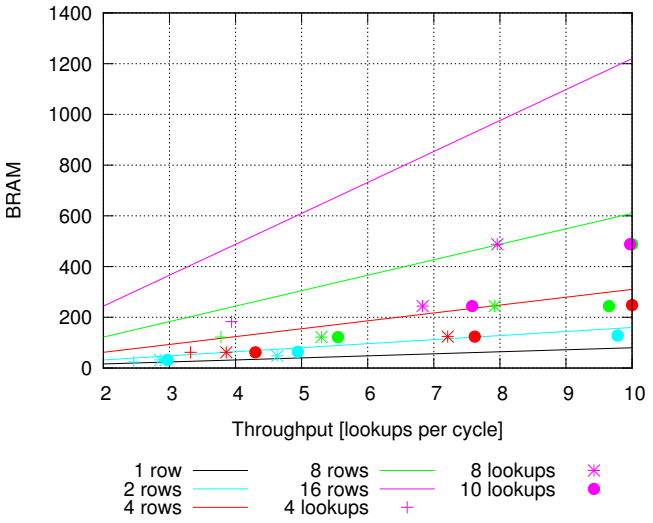

Fig. 4. The relation between memory and throughput for 4 hash functions.

more chance for the lookups to be better spread out between different rows, thus the probability of conflicts decreases. In case of 16 BlockRAM rows (pink), it is possible to achieve nearly twice the throughput without any memory duplication even when using the architecture with only 4 lookups (cross). If we use architectures with 8 (star) or 10 (full circle) lookups the speedup is even further amplified and nearly 7 or 7.5 times higher throughput can be achieved with no additional memory requirements. Additionally, with two times replicated memory, we can achieve nearly the full throughput of 10 lookups per cycle. This means that we can achieve 99.7 % of throughput with only 40 % of used memory.

The number of hash functions has no effect on the efficiency of the proposed optimization approach (only on the efficiency of cuckoo hashing itself). This can be clearly seen by comparing figure 3 with figure 4. Figure 4 shows the relation of utilized memory and achieved throughput for different architecture configurations with 4 hash functions. Graphs shown by figures 3 and 4 are pretty much the same only shifted slightly along the y-axis. The increase in memory requirements is offset by the higher capacity of the architectures (see table I).

Figures 3 and 4 might suggest that architectures with more lookups (inputs) are always better. However, this is not the case when it comes to utilized on-chip logic resources. Architecture with more lookups needs more hash function computations, more buffers, and larger distributors. The relation between on-chip logic, more specifically required LUTs, and throughput for 3 hash functions is illustrated by figure 5. The graph shows that if we use an architecture with for example 10 lookups (pink) the logic requirements go up together with the level of memory duplication and the achieved throughput. Memory-optimized architecture with 10 lookups, 16 rows and 4 memory ports (two memory replicas) achieves 99.7 % of throughput requiring only 40 % of memory at a cost of 466 % of LUTs compared to the simple approach with 10 lookups

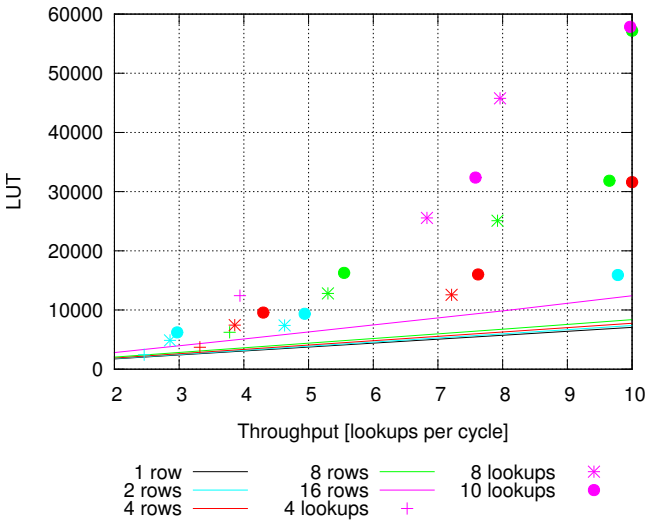

Fig. 5. The relation between used logic and throughput for 3 hash functions.

and 16 rows. From a different point of view, the optimized architecture with 10 lookups, 2 rows and 2 memory ports (no replication) achieves 48.5 % increased throughput requiring the same memory at a cost of 244 % increase in LUTs compared to the simple approach with 2 lookups and 2 rows. However, we argue that the decreased memory requirements or increased throughput, depending on the way we look at it, is a favorable trade-off for the increase in on-chip logic. In many cases, even the increased logic requirements are still feasible for current FPGAs (only a few percents of the total available), while increasing the throughput without the need to replicate memories can prove to be more critical.

In order to better illustrate the impact of the proposed memory optimization on the achieved results, we analyze them from different views. First of all, let's take a look at the best results that we can achieve if we want to reach a given minimal

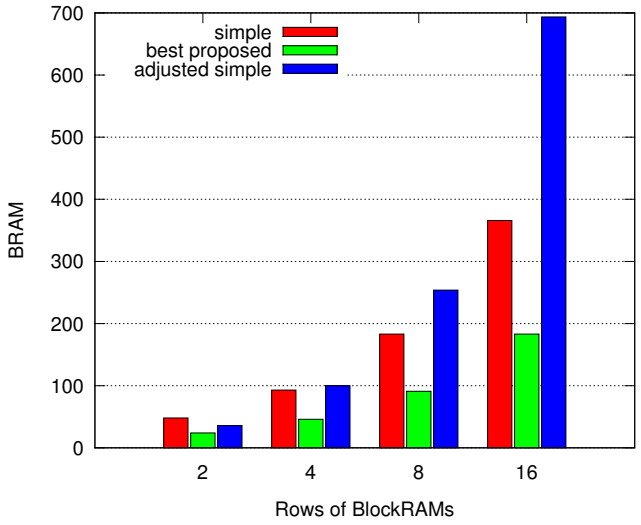

Fig. 6. Memory requirements comparison when achieving at least 800 Gbps.

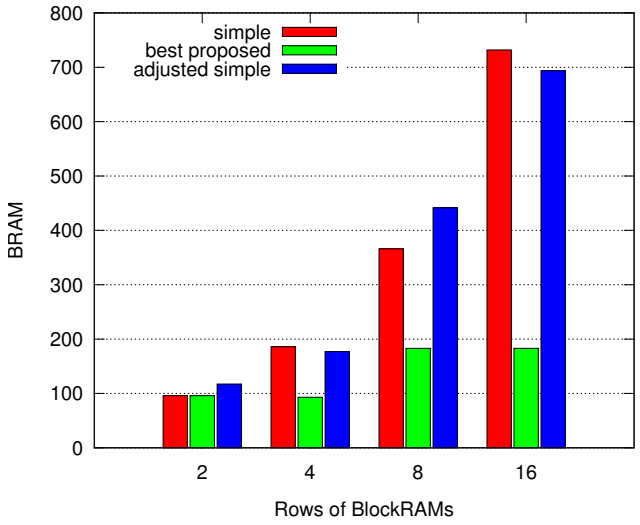

Fig. 7. Memory requirements comparison when achieving at least 1.6 Tbps.

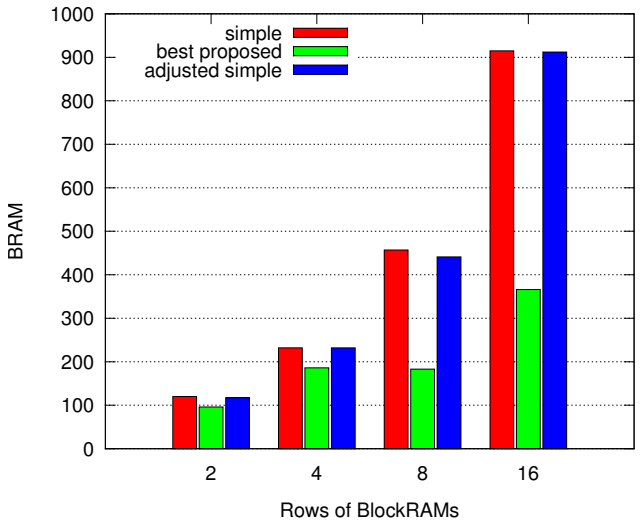

Fig. 8. Memory requirements comparison when achieving at least 2.4 Tbps.

throughput. Figures 6, 7, 8 illustrate the memory requirements of the best configurations of the proposed approach (green) compared to the baseline given by the simple memory replication (red) when we want a throughput of 800 Gbps, 1.6 Tbps or 2.4 Tbps. The best configuration is the one that requires the least memory while still satisfying the minimal throughput threshold. This obviously means that actually achievable throughputs of compared simple and optimized configurations are not the same. For better comparison, we can leverage the fact that memory of the simple approach scales linearly with throughput and adjust the required memory to the point where the simple approach has exactly the same throughput as the optimized (blue). We can see that our approach is more and more effective as the total capacity of the cuckoo hash table rises. For 2 rows it is possible to achieve the same throughput as simple replication with somewhere between 67 % and 80 % of required memory (after adjustment), while for 16 rows only between 25 % and 40 % of memory is needed. To be more precise the most significant factor that governs how much memory can be saved is the ratio between the number of rows (capacity) and required throughput (parallel lookups). The higher the capacity the better the results become as the lookups can be spread among more rows.

On the other hand, we can analyze the achievable throughput for a given number of BlockRAMs (e.g. 200) that we have available. The best cases of the proposed optimized approach are obtained when using architectures with 32 lookups. This is chosen mainly because for 2 rows of BlockRAMs the memories can be duplicated up to 8 times in the simple solution, which means 16 lookups. Therefore to obtain reasonable results we chose architectures with at least twice as much lookups. The results are shown in figure 9. An interesting thing can be observed: even as the number of rows (and therefore capacity) increases and the duplication factor decreases the throughput of the proposed approach stays relatively the same. This is caused by the fact that while the number of copies of memories (memory access ports) decreases it is balanced by a better spreading of all the lookups between more rows of independently operating BlockRAMs. If we choose other numbers of BlockRAMs the observed trend is pretty similar.

V. Conclusion

The paper presents novel memory efficient hardware architecture for exact match packet classification at very high speeds (400 Gbps and beyond) using the cuckoo hashing algorithm. The proposed architecture offers an easily configurable tradeoff between achieved throughput, required memory, utilized logic, and rule capacity. With the proposed optimization, it is possible to implement exact match packet classification for large rulesets operating at very high throughputs with efficient utilization of available memory. There are several

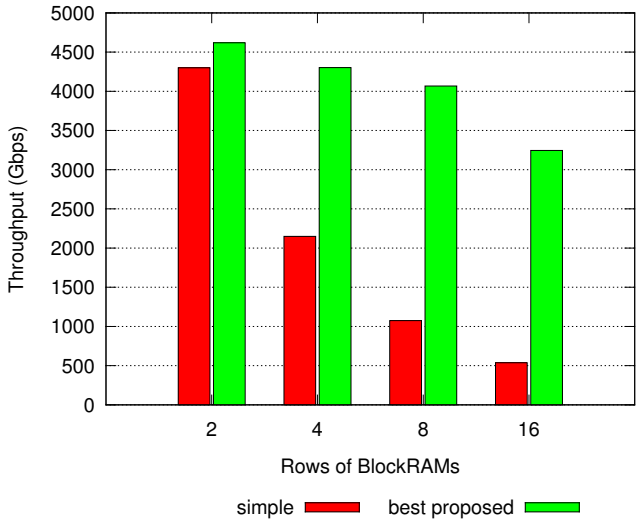

Fig. 9. Maximal throughputs of evaluated approaches for 200 BlockRAMs.

ways in which the architecture can be used – either to maximize throughput and rule capacity on devices with limited memory resources or to minimize memory requirements while satisfying needed rule capacity and throughput.

Experimental results with the proposed simple and optimized architectures of cuckoo hashing show few interesting facts. First of all the optimized architecture is considerably more memory efficient than a simple replication approach. With correct configuration, we are able to achieve 99.7 % of throughput for only 40 % of required memory compared to the simple approach. If the required rule capacity of the architecture is high enough our optimized approach is generally able to retain the same throughputs with only 25-40% of memory utilized compared to the simple solution. This way we can achieve an unprecedented throughput of 2.4 Tbps and effective capacity of over 44 000 IPv4 5-tuple (flow) rules for the cost of only 366 BRAMs. The only downside of the proposed optimized architecture is increased requirement of on-chip logic. However, we argue that the benefits of decreased memory requirements and increased throughput outweigh this issue in most practical cases.

Acknowledgments

This research is supported by the project Reg. No. CZ.02.1.01/0.0/0.0/16_013/0001797 by the MEYS of the Czech Republic; the IT4Innovations excellence in science project IT4I XS – LQ1602; and by the Ministry of the Interior of the Czech Republic projects VI20172020064 and VI20152019001.

References

[1] A. Caulfield, E. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, "A cloud-scale acceleration architecture," in *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, October 2016.

[2] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[3] P. Benáček, V. Puš, and H. Kubátová, "P4-to-VHDL: Automatic generation of 100 Gbps packet parsers," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2016, pp. 148–155.

[4] P. Benáček, V. Puš, H. Kubátová, and T. Čejka, "P4-to-VHDL: Automatic generation of high-speed input and output network blocks," *Microprocessors and Microsystems*, vol. 56, pp. 22 – 33, 2018.

[5] D. Taylor and J. Turner, "Scalable packet classification using distributed crossproducing of field labels," in *24th Annual Joint Conference of the IEEE Computer and Communications Societies*, 2005, pp. 269–280.

[6] P. Gupta and N. McKeown, "Packet classification using hierarchical intelligent cuttings," in *Proc. Hot Interconnects*, 1999.

[7] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," in *Conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2003, pp. 213–224.

[8] H. Lee, W. Jiang, and V. K. Prasanna, "Scalable High-Throughput SRAM-Based Architecture for IP Lookup Using FPGA," in *International Conference on Field Programmable Logic and Applications*, 2008.

[9] S. Dharmapurikar, H. Song, J. Turner, and J. Lockwood, "Fast packet classification using Bloom filters," in *ANCS '06: Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems*. New York, NY, USA: ACM, 2006, pp. 61–70.

[10] V. Puš and J. Kořenek, "Fast and scalable packet classification using perfect hash functions," in *FPGA '09: Proceedings of the 17th international ACM/SIGDA symposium on Field programmable gate arrays*. New York, NY, USA: ACM, 2009.

[11] J. Kořenek, V. Puš, and J. Blaho, "Memory optimization for packet classification algorithms," in *Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ser. Association for Computing Machinery. Association for Computing Machinery, 2009, pp. 165–166.

[12] H. Le and V. K. Prasanna, "Scalable Tree-based Architectures for IPv4/v6 Lookup Using Prefix Partitioning," *IEEE Trans. Comput.*, vol. 61, no. 7, pp. 1026–1039, Jul. 2012, ISSN 0018-9340.

[13] Y. Qi, J. Fong, W. Jiang, B. Xu, J. Li, and V. Prasanna, "Multi-dimensional packet classification on fpga: 100 gbps and beyond," in *2010 International Conference on Field-Programmable Technology*, Dec 2010, pp. 241–248.

[14] T. V. Lakshman and D. Stiliadis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching," *SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 4, pp. 203–214, 1998.

[15] H. Song and J. W. Lockwood, "Efficient packet classification for network intrusion detection using FPGA," in *FPGA '05: Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*. New York, NY, USA: ACM, 2005, pp. 238–245.

[16] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and scalable layer four switching," *SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 4, pp. 191–202, 1998.

[17] W. Jiang and V. K. Prasanna, "Scalable packet classification on FPGA," *IEEE Transactions on VLSI Systems*, vol. 20, no. 9, September 2012.

[18] M. Kekely and J. Korenek, "Packet classification with limited memory resources," in *2017 Euromicro Conference on Digital System Design*. Institute of Electrical and Electronics Engineers, 2017, pp. 179–183.

[19] R. Pagh and F. F. Rodler, "Cuckoo hashing," in *Algorithms - ESA 2001*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2001, vol. 2161, pp. 121–133.

[20] A. Kirsch, M. Mitzenmacher, Y. Baohua, X. Yibo, and L. Jun, "Using a queue to de-amortize cuckoo hashing in hardware," 2007. [Online]. Available: http://www.eecs.harvard.edu/~michaelm/postscripts/aller2007.pdf

[21] L. Kekely, M. Žádník, J. Matoušek, and J. Kořenek, "Fast lookup for dynamic packet filtering in FPGA," in *17th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*. Warszaw, Poland: IEEE Computer Society, 2014, pp. 219–222, iSBN: 978-1-4799-4558-0.

[22] L. Kekely, J. Kucera, V. Pus, J. Korenek, and A. V. Vasilakos, "Software defined monitoring of application protocols," *IEEE Trans. Comput.*, vol. 65, no. 2, pp. 615–626, Feb. 2016.

[23] Xilinx, *UltraScale and UltraScale+ FPGAs Packaging and Pinouts*, Xilinx Inc., 2016, UG575.