Hash-based OpenFlow Packet Classification on Heterogeneous System Architecture

Yeim-Kuan Chang and Tung-Yin Chi
Department of Computer Science and Information Engineering
National Cheng Kung University,
Tainan City, Taiwan, R.O.C.
ykchang@mail.ncku.edu.tw

***Abstract*—** Packet classification is a very important component for today's network architecture. It can help or provide packet forwarding and other network functions. With the development of Internet and the emergence of software-defined networking (SDN), the methods designed for the traditional 5-dimensional rule set is not sufficient to process the current rule set that contains rules of 12 or more dimensions. The main problem is to process the rule sets of 12 or more dimensions in high throughput. To achieve high throughput, we study the implementations on GPU where some use a single hash table and others use Binary Range Tree to process the searching. In 12-dimensional rule sets defined by OpenFlow 1.0, 8 fields are in the format of exact value or wildcard, and so using the single hash table or binary range tree is not efficient. Another problem to implement packet classification on GPU is that we must transfer the input data and results via the PCI-E bus that will incur long bus latency. In this paper, we propose a modified hash table to process the fields that contain only exact value or wildcard, and use the compressing method to reduce memory consumption. On the other hand, we implement the proposed method on APU that uses Heterogeneous System Architecture to skip the bus latency. According to the experimental results on AMD A10-7850 APU, our method can achieve the throughput of 1814 MPPS, and can support the rule sets of more than 12K 12-dimensional rules. The achieved throughput is 10 times of the methods on legacy GPU.

Keyword: Packet Classification; OpenFlow; APU; GPU; Hash Table

I. INTRODUCTION

The responsible device for forwarding packets over networks is called router and switch. These devices will fetch the header information of the packets, and analyze the information to determine where or which port to send out these packets. These actions will be repeated until the packet have reached its destination node. Today's routers and switches support not only forwarding packets, but also Quality of Service (QoS), firewall, traffic control, or virtual private network (VPN) and more applications.

Packet classification is a method to determine what actions to be applied on the input packets where actions are predefined in the rule sets. The input packet may match one or more rules. We select the rule with highest priority to apply its actions on the input packets.

Several differences between traditional 5-dimensional packet classification and OpenFlow packet classification are mainly in the number of fields in a rule and the type of fields. Eight of the 12 fields in OpenFlow 1.0 are exact values or wildcard and so using the traditional packet classification methods to process these fields is not efficient.

We proposed a scheme to speed up the fields processing of exact values. The scheme can be used as an accelerator for traditional packet classification method. By combining this scheme and traditional packet classification method, the switches can achieve very high speed for both traditional and OpenFlow rules.

The rest of this paper is organized as follows. In section II, we introduce work related to packet classification. Section III introduces the proposed scheme that is 3-Layer Hash Tree, this is a Decision-tree based algorithm, and add hashing method to process the fields with exact values. The experimental simulation and comparisons with many existing state-of-art implementatons are shown in section IV. Section V concludes the paper.

II. RELATED WORK

A. OpenFlow Rule Table

OpenFlow switches contain a set of flow tables, and each flow table contains a set of rules. The rule sets are used to classify the packets. A rule is said to be a match if all the fields of the rule match the header values of the incoming packet.

In OpenFlow version 1.0, the flow table contains 12 fields including 5 traditional packet classification fields that are source/destination IP addresses (32), source/destination transport ports (16), and IP protocol (8) and 7 additional fields that are ingress port (6), MAC source/destination addresses (48), Ethernet type (16), VLAN ID (12), VLAN priority (3), and IP ToS (6), where the numbers in parentheses are the bit counts. Totally, each entry needs 243 bits. The source/destination IP address fields are of type prefix, the source/destination transport port fields are of type range, and other 8 fields are of type singleton. The current version of OpenFlow is 1.5.1 [23]. However, version 1.6 has been available since September 2016, but accessible only to ONF's members. The proposed hash based scheme used OpenFlow 1.0 as a test example and can be easily extended to later versions.

B. Bloom Filter

Bloom Filter first described by Bloom in 1970 is a high space-efficient data structure. It can be used to check if an element is in a set or not. It has high space efficiency and constant lookup time, both are much better than normal algorithm. It may has false positive matches, but does not have false negatives. Another disadvantage of Bloom filter is that it is difficult to delete an element.

If we want to check an element is in a set or not, the simplest method is storing all the elements in the set, and compare all elements to check. We can use Tree, Linked-list or hash table to implement this method. When the number of elements increases, the memory consumption also increases, and so the speed of searching will slow down.

The basic idea of Bloom Filter is, when an element is added in to a set, we use N hash functions to generate N locations in a bitmap, and then set all of them to one. When searching the element, we just need to check whether all the corresponding locations are one or not. If they are all ones, it is likely that the element exists in the set. Otherwise, if one or more of the locations are zeros, it is impossible that the element exist in the set. Therefore, we can use Bloom filter as a pre-classifier.

C. *Accelerated Processing Unit (APU)*

In the legacy x86 PC architecture, Central Processing Unit (CPU) and Graphics Processing Unit (GPU) are independent, the communication between them goes through PCI Express bus. Advanced Micro Devices (AMD) published a new type processor named Accelerated Processing Unit (APU) that combines CPU and GPU into a single chip and share the main memory.

In the legacy architecture, when we use GPU to perform the computing, we need to copy the input data from the main memory to the GPU memory, and copy the results back from GPU memory to main memory. By the experimental results, we can see that the action of copying the data back and forth between main memory and GPU memory is the main bottleneck for the packet classification problem.

D. *Heterogeneous System Architecture (HSA)*

Heterogeneous System Architecture (HSA) is an architecture defined and developed by HSA foundation, formed by vendors like AMD, ARM, Qualcomm, Samsung, and MTK. HSA is an architecture in which CPU and GPU share the bus, memory and process. The original target is to resolve the problem of the communication latency between GPU, CPU and other processors. It can also provide better compatibility between different hardware platform (better than CUDA and OpenCL). Heterogeneous computing has been widely used in system-on-chip devices, such as video game consoles. For example, Sony PlayStation 4 uses the customized AMD APU, the CPU and GPU on the APU share the 8GB GDDR5 memory.

This architecture is first used in the Cell Broadband Engine Architecture. Heterogeneous System Architecture standard contains not only the CPUs and GPUs, it can also uses digital signal processors (DSPs), application-specific integrated circuits (ASICs) or Field-programmable gate array (FPGA) to build the system. This architecture allows any type of accelerator to operate at the same level as the CPU.

To achieve the feature that all processors can share the memory space, HSA standard defines a unified virtual address space (VAS) for the processors. On the APU without HSA, the GPU has its own memory space separated from main memory, if we want to support HSA, we need to share the page tables between GPU and CPU, then the GPU and CPU can transfer data by sharing pointers. For AMD APU, this feature is achieved by I/O memory management units (IOMMU).

As of June 2016, on x86 PC platform, only AMD’s “Kaveri” APU's and Carrizo APU's can support HAS and on other platform, the customized APU of Sony’s PlayStation 4 and ARM's “Bifrost” Mali GPU can support HSA. Except supported by hardware, the HSA needs to be supported by the operating system and device driver. For example, if we use the AMP “Kaveri” APU, it supports HSA, but we also need to install kernel module, driver, and runtime software to run the HSA program.

III. PROPOSED SCHEME

The proposed scheme is called 3-Layer Hash Tree that is a Decision-tree based data structure extended by adding hashing to process the fields with exact values. Because these fields contains only exact value or wildcard, we can simply group the rules with same type into a group, and each group has its own hash table. To determine how many hash tables to store the information of rules, we must calculate the type number first. Because the rules contains only exact value or wildcard, it means we need 2^n hash tables if the layer contains n fields. For example, in our proposed scheme, Layer 1 contains two fields (destination MAC address and source MAC address), so we need 2^2= 4 hash tables to store the rule information. Layer 3 contains 3 fields, ingress port, Ethernet type and VLAN ID and so we need 2^3=8 hash tables to store rule information.

To speed up the searching in 3-Layer Hash Tree, we use a Bloom filter phase to detect the tables which contains no target information, and so we can skip the searching operations for these tables to speed up the classification.

The L1 node consists of a Bloom filter phase and 4 hash tables. This Bloom filter phase is used as a pre-classifier for L1 hash tables. The hash tables use source MAC and destination MAC to build the hash tables. Similarly, the L2 node consists of a Bloom filter phase and 8 hash tables, where ingress port, Ethernet type and VLAN ID fields are used to build the hash tables. This Bloom filter phase is used as a pre-classifier for L2 hash tables. In the L3 node, there is a hash table with perfect hashing function. The hash tables use 2 fields to hash, VLAN priority and IP ToS.

We also use Cache table to speed up the process for the packets that have been received before. Searching the cache that only needs one memory access is much faster than searching the 3-Layer Hash Tree.

In the OpenFlow rule table, there are two types of rules, microflow and macroflow. Microflow is the rules that have exact values in all 12 fields. However, macroflow rules may contain non-exact values in some fields. Using a single hash table to process microflow rules is more efficient than using the

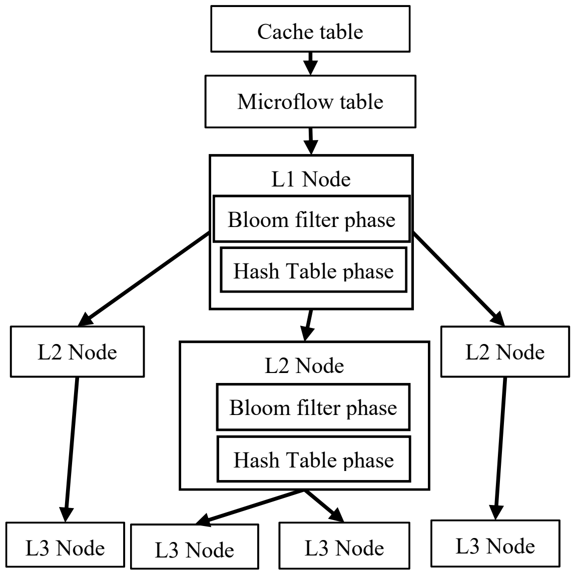

Figure 1. The 3-Layer Hash Tree.

3-Layer Hash Tree, so we split the rules into two parts, one contains microflow rules and another one contains macroflow rules. We use a single table to process the microflow table and use the 3-Layer Hash Tree to process the macroflow table.

A. Bloom Filter and Possibility Bitmap

The Bloom filter phase as shown in Figure 2 contains a main bloom filter, two 4-bit possibility bitmaps and a 4-bit ignoring flag. We use 2 hash functions based on field 11 (3-bit VLAN priority) and field 12 (6-bit IP ToS) as input data.

When both the bit positions in main bloom filter computed by these 2 hash functions are 1, there must exist at least one rule that will match the corresponding header values of the incoming packet. Then, we retrieve two 4-bit possibility bitmaps.

Assume the bit positions computed by two hash functions are 0 and 1023, respectively and both bit positions are 1. Then we will get two 4-bit bitmaps, 1111 and 0001 that will be ANDed to obtain the final result, 0001. The reason that the bitmap is 4 bits comes from the rule grouping scheme based on source and destination MAC addresses of the rules as follows. The rules with source and destination MAC addresses being both equal to wildcard and wildcard, wildcard and exact value, exact value and wildcard, and exact value and exact value are put in group 0, 1, 2, 3, respectively. As a result, if the final 4-bit bitmap is 0001, only the group 3 needs to be search. The rules in groups 1, 2, or 3 are further divided into 2^{16} subgroups based on source and destination MAC addresses by using an array of 2^{16} pointers pointing to the next layer structure (L2 node) of these subgroups.

The function of these three hash tables is 16-bit XOR folding that divides the 48-bit MAC address into three 16-bit sub-addresses and performs XOR operation. For example, if we get a hashing result 32,768, then we know that we have to store the rule in the L2 node pointed to by the 32768th pointer of the hash table.

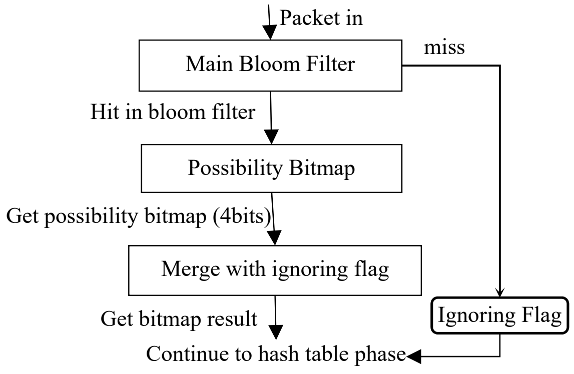

Figure 2. Bloom filter phase.

The 4-bit ignoring flag records which tables contain the rules with wildcards both in field 11 and 12 because the main Bloom filter is useless for this kind of rules. If the *i*th bit of the 4-bit ignoring flag is set to 1, it means the *i*th table contains the rule that field 11 and 12 are both wildcard, and we must search this table by ignoring the bitmap computed by the main Bloom filter and possibility bitmaps.

The bloom filter and hash tables in Layer 2 are also based on VLAN priority and IP ToS, which is similar to Layer 1. But the possibility bitmap and ignoring flag are 8 bits because fields 6-bit ingress port, 16-bit Ethernet type and 12-bit VLAN ID are used for creating eight hash tables.

We use 3 bits to represent which field value of a rule is wildcard by mapping bit 2 to ingress port field, bit 1 to Ethernet type field, and bit 0 to VLAN ID field. For example, if a rule has exact values in the fields of ingress port and Ethernet type and wildcards in VLAN ID field, it belongs to group 6 (110).

The size of hash table is 4096, and each entry is a pointer that points to an L3 node. The hash function uses a 12-bit XOR folding hashing function based on ingress port, Ethernet type, and VLAN priority.

In L3 node, we have a hash table with perfect hashing table (direct expansion) of size 512. Each entry stores a pointer to a bucket of rules. If there are wildcards in field 11 or field 12, we must duplicate the rule into multiple buckets.

B. Compress L2 hash table

Because the L2 tables waste too many entries to store null pointers, it uses about 1GB memory, about 97% of total memory consumption of the 3-Layer Hash Tree, so we propose a method to compress L2 tables.

The basic idea is to store only the pointers that point to exist L3 nodes, and remove the null pointers. We use an array of size *k* pointers where *k* is the number of non-NULL pointers in the original hash table. We also need a 4096-bit bitmap to record the positions of original hash table corresponding to those non-NULL pointers.

C. Cache and MicroFlow table

According to the experimental results of [17], 35 percent of flows contains 95 percent of packets. In other words, we can store the 35% flows in one single hash table, and this hash table can process 95% of input packets without searching the 3-Layer

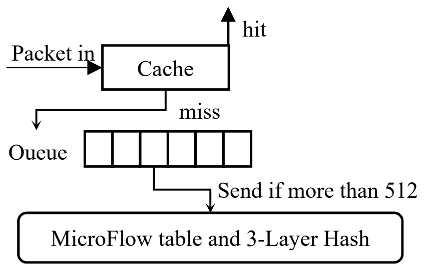

Figure 3 Optimization Model

Hash Tree. This hash table is called "Cache" and it can reduce a lot of packet classification time. In our scheme, the size of cache hash table is 1024, and use first in first out (FIFO) as cache replacement algorithm. We use this cache to classify the packets, if the packets are hit in the cache, we directly return the result. If the packets are miss in the cache, we push these into a queue of the 3-Layer Hash Tree, once the number of packets in the queue is equal or more than 512, we send first 512 packets of the queue into the GPU to search for result. The batch size is set to 512 because the limit of parallel threads on the APU is 512.

MicroFlow is a type of flow rules that have all fields with exact value. So, we can use a simple hash table to store and classify this type of rules, and this method also only needs one memory access to search the target entry. In our scheme, the maximum size of microFlow table is 65,536.

On the other hand, because of the high collision rate of traditional hashing method, we use the Cuckoo Hashing to replace traditional hashing method. With the Cuckoo Hashing, we can reduce the collision rate to less than 0.01%. In our experiment, we use a microflow table of 50K rules, and no any collision was founded.

D. Utilization of Stream Processors (SP Optmization)

If we send a batch of headers into APU, some will hit in cache and others will miss in cache. This situation will cause some of Stream Processors idle and waste the computing resource. Therefore, we split the cache and 3-Layer Hash Tree into two parts and process them in two different threads. In first part, we will only search the cache, and push the packets that miss in cache into a queue, once the number of packets in the queue is equal to or more than 512, we send the first 512 packets into the second part. The second part will search the microFlow table and 3-Layer Hash Tree as shown in Figure 3.

E. Update data structure of Bloom filter phase

In the Bloom filter phase of our scheme, it contains main bloom filter and possibility bitmaps where the possibility bitmap is a type of modified bloom filter. The original bloom filter is easy to insert, but it is impossible to delete. If we force to delete and set the target bits to 0, it will cause false negatives. To resolve the problem of deletion, we use counting bloom filter to replace the original bloom filter. The main idea of counting bloom filter is, for each bit in bloom filter, we add a counter for the bit, when inserting a new element, we set the target bit to 1 as in the original bloom filter and also increase the counters of these bits. When deleting an existing element, we decrease the counter first. Then, we check the counter. If the value of counter is 0, we set the bit position to 0. As a result, we can delete element in bloom filter without causing the false negative.

In our scheme, if we use the counting bloom filter to replace the traditional bloom filter, we use 8-bit counters in L1, 4-bit counters in L2. Overall, we need 1.125 KB to store the counters in each L1 node, 4.25 KB to store counters in each L2 node. With the 12K rule set, these counters need 38.84 MB, the total memory consumption of our scheme is 76.84 MB.

IV. EXPERIMENTAL RESULTS

To share the code on different platforms, we use C++ AMP to program the proposed schemes. This APU contains 4 cores CPU and 8 GPU Radeon cores, and the GPU contains 512 stream processors (SP), and it can run 512 threads concurrently. The APU supports HSA (Heterogeneous System Architecture) where CPU and GPU in this APU can share the main memory and GPU can directly access the memory space of CPU. We use another platform with legacy GPU to compare with APU.

We use FRuG [7] to generate two types of 12-dimensional rule sets, the microflow table that contains only the rules without wildcard values and the macroflow table that contains 127 types of rules with wildcards (*) values. The size of microflow table is fixed at 50K while the size of macroflow table is 12K, All the parameters used for this generator is set as default. Because we cannot get the rule sets used in the real networks, we first randomly generate the rule set containing rules without wildcard, and generate the 12K rule set contains 127 types with 100 rules for each type randomly.

We evaluate the performance of our proposed scheme on the following two platforms. And we use 3 different trace files with different cache hit rate, 0%, 50% and 100%.

In platform APU, we use AMD A10-7850 APU containing 4-core 3.8GHz CPU, 512-SP 850MHz GPU, and 16GB DDR3-1600 RAM. Also, OS is Ubuntu 14.04 and Compiler is HCC 0.8. In platform Legacy GPU, we use Intel i5-4460 containing 4-core 3.2GHz CPU, NVIDIA GTX-750 512-SP 1125MHz GPU, and 16GB DDR3-1600 RAM. Also, OS is Windows 7 and Compiler is Microsoft VC++ 2015.

Table 1 shows the throughput of our proposed scheme for the 12K rule set on APU platform. Our scheme can achieve 1,836-1,983 MPPS. On legacy GPU platform, it only can achieve 183 MPPS, about 1/10 of APU platform, but if we calculate the throughput without bus latency (memory copying time), the throughput is near to APU. This can prove that bus latency is a bottleneck of using GPU for packet classification processing.

As shown in Table 2, the results are the memory consumption of our proposed scheme. The 3-Layer Hash Tree without any optimization needs 1,190 MB to store the data structure. The 3-Layer Hash Tree with compressing L2 tables needs only 70 MB. Table 3 shows the number of nodes in each layer.

Table 1. Throughputs with different optimizations for 12K rules.

Platform	Optimization	Cache hit rate of trace	Throughput (MPPS)
APU	None	0%	1,586
APU	BF and cache	0%	1,836
APU	BF and cache	50%	1,854
APU	BF and cache	100%	1,983
APU	BF, cache, SP Opt	0%	1,836
APU	BF, cache, SP Opt	50%	1,899
APU	BF, cache, SP Opt	100%	1,983
Legacy GPU	None	0%	183
Legacy GPU (no memory copying time)	None	0%	1,433

Comparing the non-optimized method and compressing L2 tables, we can reduce 94.11% of memory consumption, and it will not decrease too much throughput, comparing to the non-optimized method, it can achieve 90% throughput of non-optimized method, only decrease 10% of throughput.

Table 4 shows the performance in terms of cost and power consumption for APU and legacy GPU platforms. The cost of APU platform A10-7850 is only $115 which is much lower than the legacy GPU platform NVIDIA Tesla K40 Graphic Card (~$3000) [6]. The power efficiency of our platform is also better, where the thermal design power (TDP) of NVIDIA Tesla K40 Graphic Card is 235 watts and the TDP of A10-7850 is 95 watts. In our experiment, the power consumption of whole APU PC is about 130 watts when the loading is full.

As shown in Table 5, we compare our proposed scheme with other packet classification schemes implemented on various hardwares. On FPGA platforms, the proposed scheme of [1] is decision-tree based method. [2] and [3] are decomposition based, while [2] uses the hash-based merging and [3] uses the BV-based merging. In these 3 methods, only the [1] can use 12-dimensional rule sets. On general purpose multi-core (CPU) platforms, [4] is decision-tree based and [5] is decomposition based while [4] cannot use OpenFlow rule sets and [5] can use 15-dimensional rule sets. On GPU platforms, the scheme of [6] is decomposition based and it uses 12 binary range trees to process the 12-dimensional rule sets. The scheme in [20] is a decision-tree based method and it can support 100K 15-dimensional ruleset. We can see that the proposed scheme implemented on APU outperforms all other existing schemes.

V. CONCLUSION

In this paper, we proposed a high throughput scheme. This is a decision-tree based method and use the hash-based method to assist the processing, it can achieve high throughput on APU platform. To resolve the memory consumption problem, we proposed a compressing method to resolve this problem. To skip the unnecessary searching actions, we use the Bloom filter method as a pre-classifier. This scheme can process the rule set

Table 2 Memory and throughput without optimizationd for 12K rules.

Method	Optimization	Memory (MB)	Throughput (MPPS)
3 Layer Hash Tree	None	1,190	1,586
3 Layer Hash Tree	Compress L2 table	70	1,428

Table 3 Number of nodes in each layer for 12K rules.

	Layer 1	Layer 2	Layer 3
Layer 1	1	959	1,171

Table 4. Cost and power on different platforms.

Type	Name	Peak Gflops	Stream Processors
GPU	NVIDIA Tesla K40	4,290	2,880
GPU	NVIDIA Tesla K20	3,520	2,496
APU	AMD A10-7850	737 (GPU) 118 (CPU)	512

Type	Name	Clock rate (MHz)	Thermal design Power (Watts)	Price (USD)
GPU	NVIDIA Tesla K40	745	235	3000
GPU	NVIDIA Tesla K20	706	225	2395
APU	AMD A10-7850	825	95	115

with larger size than other method ([1] and [6]). If the bugs of HCC compiler can be resolved, we can utilize the larger rule sets and achieve higher throughput, before the bugs are resolved, our scheme already can achieve higher throughput than others ([1], [2], [3], [4], [5] and [6]). On the other hand, the hardware platform of our scheme has lower price and lower power consumption than other GPU like platform (lower than [6]).

By utilizing the APU platform and a 12K rule set generated by FRuG [7], the best throughput of experimental result of our scheme can achieve 1836 MPPS without cache, and 1983 MPPS with cache. The memory consumption without any optimizations but with L2 table compression is only 70 MB with a mild throughput degradation to 1428 MPPS. The power consumption of whole platform is about 115 watts.

REFERENCES

[1] W. Jiang and V. K. Prasanna, “Scalable Packet Classification on FPGA,” IEEE Trans. VLSI Syst., vol. 20, no. 9, pp. 1668–1680, 2012.

[2] V. Pus, J. Korenek, and J. Korenek, “Fast and Scalable Packet Classification using Perfect Hash Functions,” in Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays (FPGA), 2009, pp. 229–236.

[3] T. Ganegedara and V. K. Prasanna, “StrideBV: Single Chip 400G+ Packet Classification,” in 13th IEEE International Conference on High Performance Switching and Routing (HPSR), 2012, pp. 1–6.

[4] Y. Ma, S. Banerjee, S. Lu, and C. Estan, “Leveraging Parallelism for Multi-dimensional Packet Classification on Software Routers,” SIGMETRICS Perform. Eval. Rev., vol. 38, no. 1, 2010, pp. 227–238.

[5] Yun Qu, Shijie Zhou, Viktor K. Prasanna, “Scalable Many-Field Packet Classification on Multi-core Processors,” in 25th International Symposium on Computer Architecture and High Performance Computing, 2013, pp. 33-40.

Table 5. Performance comparison with various types of platforms.

Platform	Method	Hardware	Throughput (MPPS)	# of Rules	# of Fields
FPGA	Scalable Packet Classification on FPGA [1]	Virtex-5	125	1K	12
	Fast and Scalable Packet Classification using Perfect Hash Functions [2]	Virtex-5	312	1K	5
	StrideBV: Single Chip 400G+ Packet Classification [3]	Virtex-3	1250	1K	5
General purpose multi-core processor	Leveraging Parallelism for Multi-dimensional Packet Classification on Software Routers [4]	Intel Xeon X5550 (4 cores @ 2.66GHZ)	46	30K	5
	Scalable Many-field Packet Classification on Multi-core Processors [5]	AMD Operation 6278 x2 (16 cores @ 2.4GHZ)	30	32K	15
GPU	High-Performance Packet Classification on GPU [6]	NVIDIA Tesla K40 (2880 SPs @ 745MHz)	44.1 (TCAM)	4K	12
	Many-Field Packet Classification for Software-Defined Networking Switches [20]	NVIDIA Tesla K20C (2496 SPs @ 706MHz)	170	10K	15
	3 Layer Hash Tree (Proposed scheme)	NVIDIA GTX-750 (512 SPs @ 1125MHz)	183	12K	12
APU	3 Layer Hash Tree (Proposed scheme)	AMD A10-7850 (512 SPs @ 825MHz)	1836	12K	12

[6] Shijie Zhou, Shreyas G. Singapura, Viktor K. Prasanna, “High-performance packet classification on GPU,” High Performance Extreme Computing Conference (HPEC), 2014, pp. 1-6.

[7] T. Ganegedara, W. Jiang, and V. Prasanna, "Frug: A benchmark for packet forwarding in future networks," in Proc. IPCCC '10, 2010.

[8] OpenFlow Foundation, “OpenFlow Switch Specification Version 1.0.0.”Available: http://www.openflowswitch.org/documents/openflow-spec-v1.0.0.pdf

[9] HSA foundation, “DRAFT HSA Platform System Architecture Specification 1.1” Available: http://www.hsafoundation.com/?ddownload=5114

[10] Yun R. Qu, Hao H. Zhang, Shijie Zhou, Viktor K. Prasanna, “Optimizing many-field packet classification on FPGA, multi-core general purpose processor, and GPU,” Architectures for Networking and Communications Systems (ANCS), 2015, pp.87-98.

[11] Nobutaka Matsumoto, Michiaki Hayashi, “LightFlow: Speeding up GPU-based flow switching and facilitating maintenance of flow table,” IEEE 13th International Conference on High Performance Switching and Routing, 2012, pp.76-81.

[12] Andrei Broder, Michael Mitzenmacher, Andrei Broder I Michael Mitzenmacher, “Network Applications of Bloom Filters: A Survey,” Internet Mathematics, 2004, pp.485-509.

[13] Bin Fan, David G. Andersen, Michael Kaminsky, Michael D. Mitzenmacher, “Cuckoo Filter: Practically Better Than Bloom,” 10th ACM International on Conference on emerging Networking Experiments and Technologies, 2014, pp.75-88.

[14] Wikipedia, “Heterogeneous System Architecture, ” Available: https://en.wikipedia.org/wiki/Heterogeneous_System_Architecture

[15] Weirong Jiang, Viktor K. Prasanna, Norio Yamagaki, “Decision Forest: A Scalable Architecture for Flexible Flow Matching on FPGA,” International Conference on Field Programmable Logic and Applications, 2010, pp.394-399.

[16] Yanbiao Li, Dafang Zhang, Alex X. Liu, Jintao Zheng, “GAMT: a fast and scalable IP lookup engine for GPU-based software routers,” Proceedings of the ninth ACM/IEEE symposium on Architectures for networking and communications systems, 2013, pp.1-12.

[17] Luke McHale, Jasson Casey, Paul V. Gratz, Alex Sprintson, “Stochastic Pre-Classification for SDN Data Plane Matching,” IEEE 22nd International Conference on Network Protocols, 2014, pp.596-602.

[18] Nen-Fu Huang, Shi-Ming Zhao, Jen-Yi Pan, Chi-An Su, “A Fast IP Routing Lookup Scheme for Gigabit Switching Routers,” INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE (Volume:3), 1999, pp.1429 - 1436 vol.3

[19] Kate Gregory, Ade Miller, “C++ AMP: Accelerated Massive Parallelism with Microsoft Visual C++” 2012.

[20] Cheng-Liang Hsieh, Ning Weng, “Many-Field Packet Classification for Software-Defined Networking Switches”, ANCS '16 Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems, 2016, pp.13-24

[21] Voravit Tanyingyong, Markus Hidell, Peter Sjödin, “Using hardware classification to improve PC-based OpenFlow switching”, IEEE 12th International Conference on High Performance Switching and Routing, 2011, pp.215-22

[22] Matteo Varvelloy, Rafael Laufer, Feixiong Zhang, T.V. Lakshman, “Multi-Layer Packet Classification with Graphics Processing Units”, Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies, 2014, pp.109-1.

[23] OpenFlow Foundation, "Open Networking Foundation - OpenFlow v1.5.1" (PDF), https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf