A FPGA-based Parallel Architecture for Scalable High-Speed Packet Classification

Weirong Jiang
Ming Hsieh Department of Electrical Engineering
University of Southern California
Los Angeles, CA 90089, USA
Email: weirongj@usc.edu

Viktor K. Prasanna
Ming Hsieh Department of Electrical Engineering
University of Southern California
Los Angeles, CA 90089, USA
Email: prasanna@usc.edu

***Abstract*—Multi-field packet classification is a critical function that enables network routers to support a variety of applications such as firewall processing, Quality of Service differentiation, traffic billing, and other value added services. Explosive growth of Internet traffic requires the future packet classifiers be implemented in hardware. However, most of the existing packet classification algorithms need large amount of memory, which inhibits efficient hardware implementations. This paper exploits the modern FPGA technology and presents a partitioning-based parallel architecture for scalable and high-speed packet classification. We propose a coarse-grained independent sets algorithm and then combine it seamlessly with the cross-producting scheme. After partitioning the original rule set into several coarse-grained independent sets and applying the cross-producting scheme for the remaining rules, the memory requirement is dramatically reduced. Our FPGA implementation results show that our architecture can store 10K real-life rules in a single state-of-the-art FPGA while consuming a small amount of on-chip resources. Post place and route results show that the design sustains 90 Gbps throughput for minimum size (40 bytes) packets, which is more than twice the current backbone network link rate.**

***Keywords*-FPGA; packet classification; partitioning;**

I. Introduction

Evolution of the Internet demands next-generation routers to support a variety of network applications, such as firewall processing, Quality of Service (QoS) differentiation, virtual private networks, policy routing, traffic billing, and other value added services. In order to provide these services, the router needs to classify the packets into different categories based on a set of predefined rules, which specify the value ranges of the multiple fields in the packet header. Such a function is called *multi-field packet classification*. Due to the rapid growth of the network link rate, as well as the rule set size, multi-field packet classification has become one of the fundamental challenges to designing high speed routers. For example, the current link rate has been pushed beyond the OC-768 rate, i.e. 40 Gbps, which requires processing a packet every 8 ns in the worst case (where the packets are of minimum size i.e. 40 bytes). Such throughput is impossible to achieve using existing software-based solutions [1].

This work is supported by the United States National Science Foundation under grant No. CCF-0702784. Equipment grant from Xilinx Inc. is gratefully acknowledged.

To meet the throughput requirement, recent research in this area seeks to combine algorithmic and architectural approaches, most of which are based on ternary content addressable memories (TCAMs) [2]–[4] or various hashing schemes such as Bloom Filters [5]–[7]. However, as shown in [8]–[10], TCAMs are not scalable with respect to clock rate, power consumption, or circuit area, compared to static random access memories (SRAMs). Most of TCAM-based solutions also suffer from range expansion when converting ranges into prefixes [3], [4]. Bloom Filters have become popular due to their $O(1)$ time performance and low memory requirement. However, a secondary module is needed to resolve false positives inherent in Bloom Filters, which may be slow and can limit the overall performance [11].

On the other hand, FPGA technology has become an attractive option for implementing real-time network processing engines [4], [7], [12], due to its ability to reconfigure and massive parallelism. State-of-the-art SRAM-based FPGA devices such as Xilinx Virtex-5 [13] provide high clock rate and large amounts of on-chip dual-port memory with configurable word width. Some researchers have explored implementing existing packet classification algorithms on FPGAs to achieve high throughput [4], [7], [12]. However, few of them can support large rule sets (e.g. more than 10K rules), due to their excessive memory requirement.

Note that some partitioning-based packet classification schemes which are recently proposed [14]–[16] can achieve much lower memory consumption, while none of them has been exploited for hardware implementation. The major challenge for mapping those schemes onto hardware is to bound the number of partitions which is nondeterministic (e.g. varying from 34 to 61 for different rule sets [14]) in original schemes. To address the challenge, this paper proposes a FPGA-based parallel architecture for scalable and high-throughput packet classification. The paper makes following contributions.

- Based on the idea of the Independent Sets [14], we propose a *coarse-grained independent sets* algorithm to reduce the number of partitions at the cost of increasing the number of linear search. Such extra cost is alleviated by pipelining the search process in hardware.
- We combine the coarse-grained independent sets al-

Table I
EXAMPLE RULE SET. (SA/DA:8-BIT; SP/DP: 4-BIT; PROTOCOL: 2-BIT)

Rule	Source IP (SA)	Destination IP (DA)	Source Port (SP)	Destination Port (DP)	Protocol	Priority	Action
R1	*	*	2 – 9	6 – 11	Any	1	act0
R2	1*	0*	3 – 8	1 – 4	10	2	act0
R3	*	0111*	9 – 12	10 – 13	11	3	act1
R4	*	110*	11 – 14	4 – 8	Any	4	act2
R5	001*	11*	1 – 4	9 – 15	10	5	act2
R6	001*	11*	1 – 4	4 – 15	10	5	act1
R7	100*	00*	0 – 15	5 – 6	11	6	act3
R8	100*	00*	0 – 15	5 – 6	Any	6	act0
R9	100*	0111*	0 – 15	7 – 9	11	7	act2
R10	111*	110*	0 – 15	4 – 9	Any	7	act1

gorithm seamlessly with the cross-producting scheme [17] which is among the fastest packet classification algorithms [16]. To bound the number of partitions, we split several coarse-grained independent sets out of the original rule set and apply the cross-producting scheme for remaining rules. The algorithm for selecting rules into coarse-grained independent sets is optimized to reduce the cross-producting table size.

- To reduce the processing latency, we propose to use quadtrees for the search on each field of the packet header. The search process is pipelined to achieve high throughput. By encoding the comparison results into the memory address, the pointers to children nodes are removed so that the memory consumption is reduced.
- Implementation results show that our architecture can store 10K 5-field rules in a single Xilinx Virtex-5 FPGA. It sustains 90 Gbps throughput for minimum size (40 bytes) packets. To the best of our knowledge, our design is among few hardware implementations that are able to perform multi-field packet classification at over 40 Gbps while supporting a large rule set with 10K unique rules.

The rest of the paper is organized as follows. Section II has the problem statement, and summarizes related packet classification algorithms. Section III reviews the prior work on FPGA designs for multi-field packet classification engines. Section IV presents the parallel architecture and the set of proposed algorithms including the rule set partitioning and the quadtree search on each field. Section V evaluates the performance of the algorithms and the architecture. Section VI concludes the paper.

II. BACKGROUND

A. Problem Statement

An IP packet is usually classified based on the five fields in the packet header: the 32-bit source/destination IP addresses (denoted **SA/DA**), 16-bit source/destination port numbers (denoted **SP/DP**), and 8-bit transport layer protocol [1], [9]. Individual entries for classifying a packet are called *rules*. Each rule can have one or more fields and their associated values, a priority, and an action to be taken if matched. Different fields in a rule require different kinds of matches: prefix match for SA/DA, range match for SP/DP, and exact match for the protocol field. Table I shows a simplified example, where each rule contains match conditions for 5 fields: 8-bit source and destination addresses, 4-bit source and destination port numbers, and a 2-bit protocol value.

A packet is considered matching a rule only if it matches all the fields within that rule. A packet can match multiple rules, but only the rule with the highest priority is used to take action. This problem is called *best-match* packet classification. It is slightly different from *multi-match* packet classification [4] which requires reporting all the matching rules for a packet. These two types of problems target different network applications. The context of multi-match packet classification is in network intrusion detection systems (NIDS) [4], while our work focuses on next-generation high-speed routers.

B. Packet Classification Algorithms

Packet classification algorithms have been extensively studied in the past decade. Comprehensive surveys can be found in [1], [9]. Most of those algorithms fall into three categories: decision-tree-based, decomposition-based, and partitioning-based approaches.

Decision-tree-based algorithms (e.g. HyperCuts [18]), take the geometric view of the packet classification problem. Each rule defines a hypercube in a D-dimensional space where D is the number of header fields considered for packet classification. Each packet defines a point in this D-dimensional space. The decision tree construction algorithm employs several heuristics to cut the space recursively into smaller subspaces. Each subspace ends up with fewer rules. The cutting process is performed until the number of rules contained by a subspace is small enough to allow a low-cost linear search to find the best matching rule. Such algorithms scale well and are suitable for rule sets where the rules have little overlap with each other. However, the depth of decision tree is nondeterministic (e.g. varying from 11 to 32 for different rule sets [18]) and can be large in the worst

case. Thus decision-tree-based algorithms are usually not practical for hardware implementation.

Decomposition-based algorithms (e.g. BV [19], cross-producting [17]), perform independent search on each field and finally combine the search results from all fields. Such algorithms are desirable for hardware implementation due to their parallel search on multiple fields. However, substantial storage is usually needed to merge the single-field search results to obtain the final result [9].

Both decision-tree-based and decomposition-based algorithms suffer from $O(N^D)$ memory explosion in the worst case where N denotes the number of rules and D the number of fields in a rule [9], [16]. To reduce the memory consumption, some recent work [14]–[16] proposes to partition the original rule set into multiple subsets. The rules in each subset are mutually disjoint on one field or multiple fields, so that the search within each subset is simplified. For example, the Independent Sets algorithm [14] partitions the rule set into many independent sets where one-dimensional search is performed within each independent set. The memory requirement is thus dramatically reduced at the cost of more time needed to search multiple subsets. Although the search process on multiple independent sets can be parallelized in hardware, the nondeterministic and large number of independent sets (e.g. varying from 34 to 61 for different rule sets [14]) becomes a major challenge for hardware implementation.

III. Prior Work based on FPGAs

Most of existing FPGA implementations of packet classification engines are based on either decision-tree-based or decomposition-based algorithms. We are unaware of any FPGA design based on partitioning-based schemes.

Lakshman et al. [19] propose the Parallel Bit Vector (BV) algorithm, which is a decomposition-based algorithm targeting hardware implementation. It performs the parallel lookups on each individual field first. The lookup on each field returns a bit vector with each bit representing a rule. A bit is set if the corresponding rule is matched on this field; otherwise the bit is reset. The result of the bitwise AND operation on these bit vectors indicates the set of rules that matches a given packet. By combining TCAMs and the BV algorithm, Song et al. [4] present an architecture called BV-TCAM for *multi-match* packet classification. A TCAM performs prefix or exact match, while a multi-bit trie implemented in Tree Bitmap [20] is used for source or destination port lookup. The authors did not report the actual FPGA implementation results, though they claim that the entire circuit for 222 rules consumes less than 10% of the available logic and fewer than 20% of the available Block RAMs of a Xilinx XCV2000E FPGA. They also predict the design after pipelining can achieve 10 Gbps throughput when implemented on advanced FPGAs.

Taylor et al. [21] introduce Distributed Crossproducting of Field Labels (DCFL), which is also a decomposition-based algorithm leveraging several observations of the structure of real filter sets. Instead of using bit vectors, DCFL uses a network of efficient aggregation nodes, by employing Bloom Filters and encoding intermediate search results. As a result, the algorithm avoids the exponential increase in the time or space incurred when performing this operation in a single step. The authors predict that an optimized implementation of DCFL can provide over 100 million packets per second (MPPS) and store over 200K rules in the current generation of FPGA or ASIC without the need of external memories. However, their prediction is based on the maximum clock frequency of FPGA devices and a logic intensive approach using Bloom Filters. This approach may not be optimal for FPGA implementation due to long logic paths and large routing delays. Furthermore, the estimated number of rules is based only on the assumption of statistics similar to those of the currently available rule sets. Jedhe et al. [12] realize the DCFL architecture in their complete firewall implementation on a Xilinx Virtex 2 Pro FPGA, using a memory intensive approach, as opposed to the logic intensive one, so that on-the-fly update is feasible. They achieve a throughput of 50 MPPS, for a rule set of 128 entries. They also predict the throughput can be 24 Gbps when the design is implemented on Virtex-5 FPGAs.

Papaefstathiou et al. [6] propose a memory-efficient decomposition-based packet classification algorithm, which uses multi-level Bloom Filters to combine the search results from all fields. Their FPGA implementation, called 2sBFCE [7], shows that the design can support 4K rules in 178 Kbytes memories. However, the design takes 26 clock cycles on average to classify a packet, resulting in low throughput of 1.875 Gbps on the average. Note that both DCFL [21] and 2sBFCE [7] may suffer from false positives due to the use of Bloom Filters, as discussed in Section I.

Two recent works [22], [23] discuss several issues in implementing decision-tree-based packet classification algorithms on FPGA, with different motivations. Luo et al. [22] propose a method called *explicit range search* to allow more cuts per node than the HyperCuts algorithm. The tree height is dramatically reduced at the cost of increased memory consumption. At each internal node, a varying number of memory accesses may be needed to determine which child node to traverse, which may be infeasible for pipelining. Since the authors do not implement their design on FPGA, the actual performance results are unclear. To achieve power efficiency, Kennedy et al. [23] implement a simplified HyperCuts algorithm on an Altera Cyclone 3 FPGA. They store up to hundreds of rules in each leaf node and match them in parallel, resulting in low clock frequency (32 MHz reported in [23]). Since the search in the decision tree is not pipelined, their implementation can sustain only 0.47 Gbps in the worst cases where it takes 23

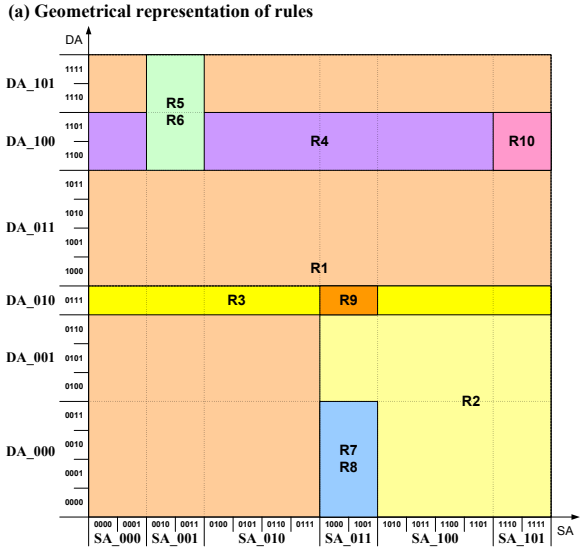
clock cycles to classify a packet for the rule set FW_20K. Jiang et al. [24] propose two optimization methods for the HyperCuts algorithm to reduce memory consumption. By deep pipelining, their FPGA implementation can achieve a throughput of 80 Gbps. However, since the pipeline depth is determined by the decision tree height, their architecture may not be practical for decision trees with various heights.

IV. Architecture and Algorithms

A. Motivations

1) Bounded Rule Set Partitioning: We observe that the partitioning-based packet classification algorithms have not been exploited for hardware implementation. The unbounded number of partitions in original schemes becomes the major challenge for mapping them onto hardware. Our work is to optimize them so that they can be implemented on FPGAs to achieve both memory efficiency and high throughput. To illustrate our idea, we use the example rule set given in Table I. If we consider only the SA and DA fields, the 10 rules can be represented geometrically on a 2-dimensional plane (Figure 1 (a)).

(a) Geometrical representation of rules



(b) Cross-product table

	SA_000	SA_001	SA_010	SA_011	SA_100	SA_101
DA_000	R1	R1	R1	R7, R8	R2	R2
DA_001	R1	R1	R1	R2	R2	R2
DA_010	R3	R3	R3	R9	R3	R3
DA_011	R1	R1	R1	R1	R1	R1
DA_100	R4	R5, R6	R4	R4	R4	R10
DA_101	R1	R5, R6	R1	R1	R1	R1

Figure 1. (a) Geometrical representation of the 10 rules in Table I; (b) Resulting cross-product table

The original cross-producting scheme [17] projects all rules onto each field, resulting in a sequence of primitive ranges for each field. A cross-product table is then built to combine the primitive ranges from all the fields. As shown in Figure 1 (a), after projecting the 10 rules onto SA and DA fields, there are 6 primitive ranges on each field. Each primitive range is labeled using the field name and the range ID. For example, "SA_000" denotes the first primitive range on the SA field. These primitive ranges from SA/DA fields result in a cross-product table with 36 entries, as shown in Figure 1 (b). To match an input packet, independent search is conducted on each field and returns the primitive range where the value of the corresponding field of the packet is matched. For instance, if there is an input packet with $SA = 10001000$ and $DA = 01110111$, the packet will match the primitive ranges SA_011 and DA_010 on SA and DA fields, respectively. Then the packet looks up the cross-product table using the indexes of SA_011 and DA_010. According to the cross-product table, rule $R9$ is matched for this packet.

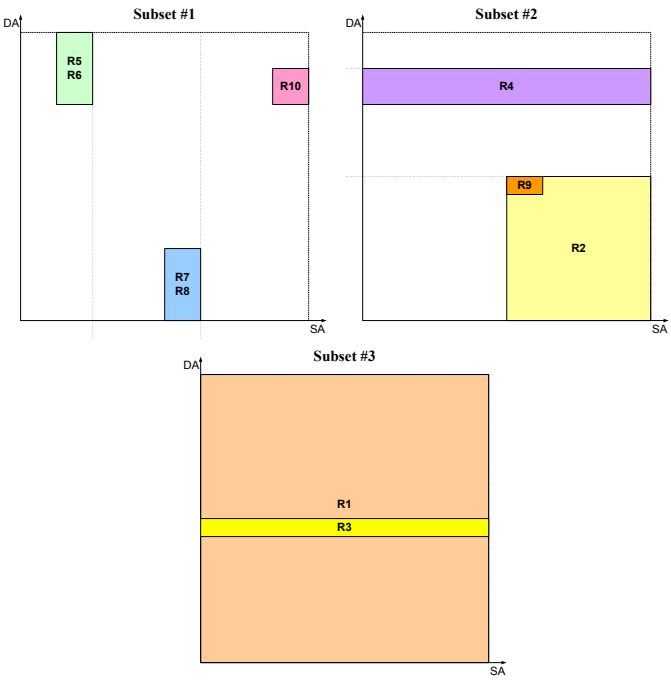


Figure 2. Partitioning the example rule set into $P = 3$ subsets. ($B = 2$)

To reduce the cross-product table size, we combine the idea from the Independent Sets algorithm [14] which requires linear memory with the number of rules. We partition the rule set into P subsets where $P - 1$ subsets are coarse-grained independent sets (explained later) and only the last subset is searched by cross-producting. Figure 2 shows the results after partitioning the example rule set into $P = 3$ subsets. The 2 rules, $R1$ and $R3$, in the third subset result in a cross-product table with 3 entries, which is much smaller than the size of the original cross-product table. Note that Wang [16] proposes a similar solution to ours,

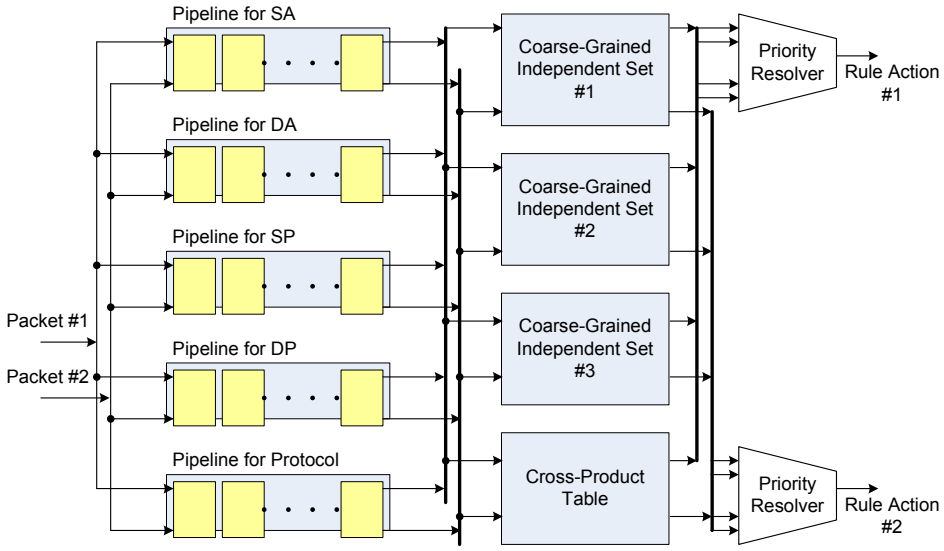


Figure 3. Block diagram of the architecture ($P = 4$)

called controlled cross-producting, which also combines the Independent Sets algorithm with the cross-producting scheme. However, the controlled cross-producting algorithm needs to convert each range into prefixes, which results in rule expansion and increases the memory consumption.

2) Coarse-Grained Independent Sets: The original Independent Sets algorithm [14] requires all the rules within an independent set must be mutually disjoint on the same field. Such a constraint limits the number of rules in an independent set and thus results in a large number of independent sets. We propose a *coarse-grained independent sets* algorithm to reduce the number of independent sets effectively. We note that the key idea behind the original Independent Sets algorithm is to divide the one-dimensional space on a field into several independent intervals [1] each of which contains only one rule. In contrast, our coarse-grained independent sets algorithm allows an independent interval to include up to B rules where B is a design-time parameter controlling the granularity of the independent sets. As shown in Figure 2, up to $B = 2$ rules can be overlapped within an independent interval, so that $R2$ and $R9$ can be put into the same subset. When $B = 1$, our coarse-grained independent sets algorithm becomes the original Independent Sets algorithm, and the first two subsets (which are coarse-grained independent sets) shown in Figure 2 have to be expanded to four subsets.

B. Architecture Overview

Figure 3 shows the block diagram of our architecture which is composed of two stages.

First, we perform the search on each individual field in parallel. Each single-field search returns the information associated with the primitive range that matches the value of the corresponding field of the input packet. The range search tree is constructed as a quadtree where each internal tree node has four children nodes. Since the quadtree is balanced, the height of the quadtree for a field containing M primitive ranges is $\log_4 M$. To achieve high throughput, we pipeline the quadtree search by assigning each tree level to a dedicated memory block. The pipeline depth (i.e. the number of stages in the pipeline) is equal to the tree height.

Second, we use several tables to combine the search results from all fields, to obtain the final matching rule. There are P tables where $P - 1$ of them are used for coarse-grained independent sets while the remaining one is for cross-producting. The table for a coarse-grained independent set is looked up using the IDs of the independent intervals based on which the coarse-grained independent set is built. For example, the table for the first coarse-grained independent set shown in Figure 2 includes 3 entries each of which has up to $B = 2$ rules. The index "01" will retrieve $R7$ and $R8$ contained in the second interval. To look up the cross-product table, the index is obtained by cascading the IDs of the primitive ranges from all the fields.

The outputs of the first stage include all information needed by the second stage. The search result from each field contains (1) the IDs of the tables to look up (2) the indices that are used for table lookup. For instance, according to Figure 1, an input packet with $SA = 10001000$ and $DA = 01110111$ will match the primitive ranges SA_011 and DA_010 on SA and DA fields, respectively. The information associated with SA_011 will include two sets of {table ID, index} tuples: {00, 01} and {10, null}. This is because SA_011 is within the "01"th independent interval of the "00"th coarse-grained independent set, as well as within the only primitive range on the SA field

[1] An interval is called independent if it has no overlap with any other interval.

of the cross-product table, as shown in Figure 2. Similarly, the information associated with DA_010 is: {01, 00} and {10, 01}, since DA_010 is within the "00"th independent interval of the "01"th coarse-grained independent set as well as within the "01"th primitive range on the DA field of the cross-product table. Then the packet uses the index "01" to look up the first table to retrieve the possibly matching rules $R7$ and $R8$, uses the index "00" to look up the second table to retrieve the possibly matching rules $R2$ and $R9$, and uses the index cascading "null" from SA and "01" from DA to look up the third table (i.e. the cross-product table) to retrieve the matching rule $R3$. Within each coarse-grained independent set, each field of the packet is matched against that of the retrieved rules to verify the matching. In this example, the packet finally matches none in the first subset, matches both $R2$ and $R9$ in the second subset, and matches $R3$ in the last subset. Since only the rule with the highest priority needs to be reported, we need a priority resolver to pick the highest-priority rule from all matching rules. In this example, $R9$ is finally returned.

To further improve the throughput, we exploit the dual-port Block RAMs provided by current FPGAs so that two packets can be input per clock cycle while the memory content is shared.

C. Partitioning Algorithm

Given P (the number of subsets), the goal of our bounded rule set partitioning scheme is to maximize the number of rules in coarse-grained indepent sets while minimizing the cross-product table size. The problem can be proved to be NP-complete using the reduction from Maximum Independent Set problem [25]. We use a greedy algorithm shown in Figure 4 to partition the original rule set into P subsets where $P-1$ of them are coarse-grained independent sets with granularity B. The last subset is reserved for the remaining rules which are not put into any of the coarse-grained independent sets and will be matched using the cross-producting scheme.

For each subset (excluding the subset for cross-producting), the algorithm chooses the field that contains the maximum number of independent intervals. Then up to B rules are selected for each independent interval. However, there can be more than B rules in an independent interval. We use a heuristic which selects the rules with the least used field specification so that the cross-product table of the remaining rules is minimized. For example, when we build the second coarse-grained independent set in Figure 2, $R2$, $R3$ and $R9$ are all within the first independent interval while $B = 2$. The field specifications of $R2$ and $R9$ are not used by other remaining rules. The removal of $R2$ and $R9$ from the original rule set can reduce the number of primitive ranges so that the cross-product table of the remaining rules has fewer entries. Thus, $R2$ and $R9$ are preferred to $R3$ whose SA field specification is same as $R1$.

Input: A set of N rules: R_o
Output: $P-1$ independent sets with granularity B: RS_i, $i = 1, 2, \cdots, P-1$

```
1:  for i ← 1 to P − 1 do
2:    d ← The field with the most number of primitive ranges
3:    // Sort right points of all primitive ranges on this field
4:    rp[N_rp] ← SortRightPoints(R_o, d)
5:    // Find the maximum coarse-grained independent set on this field
6:    prevRightPoint ← 0
7:    for j ← 1 to N_rp do
8:      currRightPoint ← rp[j]
9:      k ← 0
10:     while k < B do
11:       R_m ← the rule with the least used field specification among the rules whose dth field is within [prevRightPoint, currRightPoint]
12:       if R_m == NULL then
13:         Break
14:       end if
15:       Insert R_m into RS_i
16:       Remove R_m from R_o
17:     end while
18:     prevRightPoint ← currRightPoint
19:   end for
20:   Update the information associated with related primitive ranges
21: end for
```

Figure 4. Algorithm to partition a rule set. (N_{rp} denotes the number of right points and $rp[j]$ the jth right point.)

D. Quadtree Search on Single Fields

The first stage of our architecture is to search the primitive range on each field that matches the corresponding field of the input packet. The common method is to employ a binary search tree which needs $\log_2 M$ levels to search M ranges. In order to reduce the search latency, we propose using quadtrees where each tree node has 4 children nodes and the tree height is $\log_4 M$.

We use the right endpoints of the ranges to build a quadtree. Figure 5 (b) shows the quadtree corresponding to the 6 primitive ranges on the SA field of the example rule set shown in Figure 1. Each tree node, except the leaf nodes, contains the values of three right endpoints.

The search of a quadtree is pipelined by assigning each tree level to a separate memory block. To access the tree nodes at each level, we use the similar idea as Le et al. [26] proposed for binary search trees. Instead of explicitly storing the pointers of children nodes at each tree node, we obtain the address of the children node by cascading the address of the current node with the comparison results at

the current level. Consider the example where an input value of 8 is searched among the 6 ranges shown in Figure 5. The input value compares with the 3 values at the root node and gets the comparison result of 01. Then 01 is used as the address to access the memory in the next level. The input value continues to compare with the 3 values in the retrieved node. The comparison result is again 01. The address for accessing the memory in the third level is thus 0101.

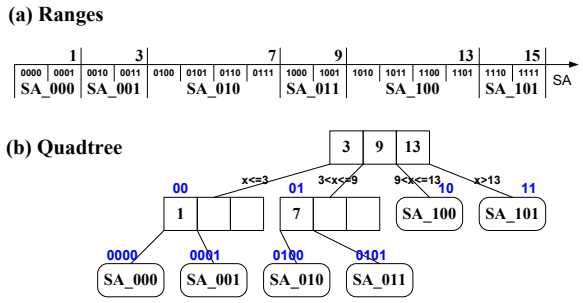


Figure 5. Building the quadtree for the SA field of the example rule set.

V. Performance Evaluation

A. Algorithm Evaluation

We evaluated the effectiveness of the proposed coarse-grained independent sets algorithm. We downloaded 4 real-life rule sets of different sizes from [27]. Their information is shown in Table II. The performance metric is the average memory size per rule which represents the scalability of the algorithm. The architecture parameters include the number of subsets (denoted P) and the granularity of the independent sets (denoted B).

First, we increased the number of subsets P while keeping $B = 1$. As shown in Figure 6 (a), partitioning the rule set into more subsets results in less memory consumption. The improvement became less significant when $P \geq 4$. We also evaluated the impact of the independent set granularity by increasing B while keeping $P = 4$. As shown in Figure 6 (b), the memory requirement was reduced dramatically when the granularity was increased.

We compared our algorithm ($P = 4$, $B = 2$) with the representative of the decision-tree-based and the decomposition-based algorithms. As Table II shows, our algorithm achieved the best scalability in terms of memory requirement.

Table II
Memory requirement (Bytes/rule) of algorithms for rule sets of various sizes

Rule sets	# of rules	Our algorithm	HyperCuts [18]	BV [19]
ACL_100	98	26.19	27.78	47
ACL_1K	916	22.78	58.10	91
ACL_5K	4415	47.61	131.21	257
ACL_10K	9603	28.55	75.46	789

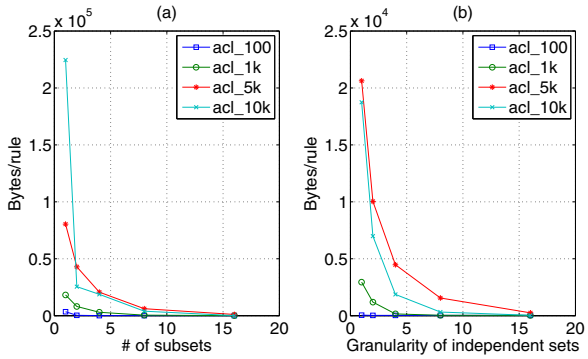


Figure 6. (a) Increasing the number of subsets ($P = 1, 2, 4, 8, 16$; $B = 1$). (b) Increasing the granularity ($B = 1, 2, 4, 8, 16$; $P = 4$).

B. Implementation Results

We implemented our design ($P = 4, B = 2$) that supported the large rule set ACL_10K using Xilinx ISE 10.1 development tools. The target device was Xilinx Virtex-5 XC5VFX200T with -2 speed grade. Post place and route results showed that our design achieved a clock frequency of 143 MHz while consuming a small amount of the on-chip resources as shown in Table III.

Table III
Resource utilization

	Used	Available	Utilization
Number of Slices	1,425	30,720	4%
Number of bonded IOBs	236	960	24%
Number of Block RAMs	96	456	21%

Table IV compares our design with the state-of-the-art FPGA-based packet classification engines. For fair comparison, the results of the compared work were scaled to Xilinx Virtex-5 platforms based on the maximum clock frequency [2]. The values in parentheses were computed based on the original data reported in those papers. Considering the time-space trade-off in various designs, we used a new performance metric, named *Efficiency*, which was defined as the throughput divided by the average memory size per rule. Our design outperformed the others with respect to throughput and efficiency.

VI. Conclusion

This paper proposes a FPGA-based parallel architecture for scalable and high-throughput packet classification. The memory efficiency is achieved by partitioning the original rule set into multiple independent sets which can be matched via one-dimensional search. We addressed the challenge for mapping partitioning-based algorithms onto hardware by extending the ideas of the independent sets and combining it

[2]The BV-TCAM paper [4] does not present the implementation result regarding the throughput. We use the predicted value given in [4].

Table IV
PERFORMANCE COMPARISON

FPGA-based engines	# of rules	Total memory (Kbytes)	Throughput (Gbps)	Efficiency (Gbps/KB)
Our approach	9603	432	91.73	2039.1
Optimized HyperCuts [24]	9603	612	80.23	1358.9
Simplified HyperCuts [23]	10000	286	10.84 (5.12)	379.0
BV-TCAM [4]	222	16	10 (N/A)	138.8
2sBFCE [7]	4000	178	2.06 (1.88)	46.3
Memory-based DCFL [12]	128	221	24 (16)	13.9

with the cross-producting scheme. Our FPGA implementation results show that our architecture can store 10K 5-field rules in a single Xilinx Virtex-5 FPGA while consuming a small amount of on-chip resources. It sustains 90 Gbps throughput for minimize size (40 bytes) packets, which is more than twice the current backbone network link rate (i.e. 40 Gbps). To the best of our knowledge, this design is among few packet classification engines realizing over 40 Gbps throughput while supporting 10K unique rules.

REFERENCES

[1] P. Gupta and N. McKeown, "Algorithms for packet classification," *IEEE Network*, vol. 15, no. 2, pp. 24–32, 2001.

[2] F. Yu, R. H. Katz, and T. V. Lakshman, "Efficient multimatch packet classification and lookup with TCAM," *IEEE Micro*, vol. 25, no. 1, pp. 50–59, 2005.

[3] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "Algorithms for advanced packet classification with ternary CAMs," in *Proc. SIGCOMM*, 2005, pp. 193–204.

[4] H. Song and J. W. Lockwood, "Efficient packet classification for network intrusion detection using FPGA," in *Proc. FPGA*, 2005, pp. 238–245.

[5] S. Dharmapurikar, H. Song, J. S. Turner, and J. W. Lockwood, "Fast packet classification using bloom filters," in *Proc. ANCS*, 2006, pp. 61–70.

[6] I. Papaefstathiou and V. Papaefstathiou, "Memory-efficient 5D packet classification at 40 Gbps," in *Proc. INFOCOM*, 2007, pp. 1370–1378.

[7] A. Nikitakis and I. Papaefstathiou, "A memory-efficient FPGA-based classification engine," in *Proc. FCCM*, 2008.

[8] F. Baboescu, S. Singh, and G. Varghese, "Packet classification for core routers: Is there an alternative to CAMs?" in *Proc. INFOCOM*, 2003.

[9] D. E. Taylor, "Survey and taxonomy of packet classification techniques," *ACM Comput. Surv.*, vol. 37, no. 3, pp. 238–275, 2005.

[10] W. Jiang, Q. Wang, and V. K. Prasanna, "Beyond TCAMs: An SRAM-based parallel multi-pipeline architecture for terabit IP lookup," in *Proc. INFOCOM*, 2008.

[11] I. Sourdis, "Designs & algorithms for packet and content inspection," Ph.D. dissertation, Delft University of Technology, 2007. [Online]. Available: http://ce.et.tudelft.nl/publicationfiles/1464_564_sourdis_phdthesis.pdf

[12] G. S. Jedhe, A. Ramamoorthy, and K. Varghese, "A scalable high throughput firewall in FPGA," in *Proc. FCCM*, 2008.

[13] Xilinx Virtex-5 FPGAs, "http://www.xilinx.com."

[14] X. Sun, S. K. Sahni, and Y. Q. Zhao, "Packet classification consuming small amount of memory," *IEEE/ACM Trans. Netw.*, vol. 13, no. 5, pp. 1135–1145, 2005.

[15] K. Zheng, Z. Liang, and Y. Ge, "Parallel packet classification via policy table pre-partitioning," in *Proc. Globecom*, 2005.

[16] P.-C. Wang, "Scalable packet classification with controlled cross-producting," *Computer Networks*, vol. 53, no. 6, pp. 821 – 834, 2009.

[17] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and Scalable Layer Four Switching," in *SIGCOMM*, 1998, pp. 191–202.

[18] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," in *Proc. SIGCOMM*, 2003, pp. 213–224.

[19] T. V. Lakshman and D. Stiliadis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching," in *Proc. SIGCOMM*, 1998, pp. 203–214.

[20] W. Eatherton, G. Varghese, and Z. Dittia, "Tree bitmap: hardware/software IP lookups with incremental updates," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, pp. 97–122, 2004.

[21] D. E. Taylor and J. S. Turner, "Scalable packet classification using distributed crossproducing of field labels," in *Proc. INFOCOM*, 2005.

[22] Y. Luo, K. Xiang, and S. Li, "Acceleration of decision tree searching for IP traffic classification," in *Proc. ANCS*, 2008.

[23] A. Kennedy, X. Wang, Z. Liu, and B. Liu, "Low power architecture for high speed packet classification," in *Proc. ANCS*, 2008.

[24] W. Jiang and V. K. Prasanna, "Large-scale wire-speed packet classification on FPGAs," in *Proc. FPGA*, 2009.

[25] J. Kleinberg and E. Tardos, *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., 2005.

[26] H. Le, W. Jiang, and V. K. Prasanna, "Scalable high-throughput SRAM-based architecture for IP-lookup using FPGA," in *Proc. FPL*, 2008.

[27] Packet Classification Filter Sets, "http://www.arl.wustl.edu/∼hs1/pclasseval.html."