POWER-EFFICIENT RANGE-MATCH-BASED PACKET CLASSIFICATION ON FPGA*

Yun R. Qu *Viktor K. Prasanna*

Ming Hsieh Department of Electrical Engineering
University of Southern California, Los Angeles, CA 90089
{yunqu, prasanna}@usc.edu

ABSTRACT

Packet classification is a kernel application performed at network routers. Many classification engines are optimized for prefix and exact match, while a range-to-prefix translation can lead to rule set expansion. Under limited power budget, it is challenging to achieve high classification throughput. In this paper, we present a high-performance and power-efficient packet classification engine on FPGA. We construct a modular Processing Element (PE); each PE compares a stride of the input packet header against a stride of a range boundary. We concatenate multiple PEs into a systolic array. Efficient power optimization techniques including self-enabled power gating and entropy-based scheduling are explored on our architecture. Experimental results show that, for 4 K 15-field rule sets, our prototype on a state-of-the-art FPGA can achieve 250 Million Packets Per Second (MPPS) throughput. Using the proposed power optimization techniques, our classification engine consumes 30% of the power without sacrificing the throughput.

1. INTRODUCTION

The development of Internet demands routers to support a variety of network applications, such as firewall processing and Quality of Service (QoS) differentiation. This makes packet classification a kernel function for network management tasks; an incoming packet can be discarded, forwarded to specific ports, or broadcast based on many criteria.

Packet classification faces the following challenges: (1) the expanding depth and width of the classification rule sets, (2) the growing complexity of the rule sets, and (3) the increasing demand for high throughput and low power. For example, in *OpenFlow* protocol [1], 15 fields of the packet header have to be examined; some fields require generic range match to be performed. Meanwhile, many emerging network applications require high throughput under constrained power budget. These factors make packet classification a critical task in high-performance routers.

Many existing solutions for packet classification employ Ternary Content Addressable Memory (TCAM) [2]. TCAM is notorious for its high cost and power consumption. State-of-the-art VLSI chips can be built with massive amount of on-chip computation and memory resources, as well as large number of I/O pins for off-chip memory accesses; FPGAs [3], with their flexibility and reconfigurability, are especially suitable for accelerating network applications.

In this paper, we propose a high-performance and power-efficient packet classification engine on FPGA. The engine can perform prefix match, exact match, or range match on any field. Efficient power optimization techniques are employed on this engine. Specifically:

- We construct a *modular PE* to match a stride of the packet header against a stride of a range boundary. We concatenate multiple PEs into a *systolic array* to sustain high clock rates for large rule sets.
- We employ a *self-enabled power gating* technique on our architecture. The modular PEs are selectively enabled to save the memory access power.
- We propose an *entropy-based scheduling* for various fields. To improve the efficiency of our power gating technique, the fields corresponding to higher entropy values are matched in the first few pipeline stages.
- We prototype our designs on a state-of-the-art FPGA. Post place-and-route results demonstrate 250 MPPS throughput while using 1.655 W power (70% reduction compared to the non-optimized designs).

The rest of the paper is organized as follows: Section 2 introduces the packet classification problem. We present our hardware architecture and optimization techniques in Section 3 and Section 4, respectively. We evaluate the performance on FPGA in Section 5. Section 6 compares our work with the related works. Section 7 concludes the paper.

2. BACKGROUND

2.1. Packet Classification

Packet classification involves classifying packets based on multiple fields in the packet header [2, 4]. The individual

*Supported by U.S. National Science Foundation under grant CCF-1320211. Equipment grant from Xilinx Inc. is gratefully acknowledged.

Table 1: An example of OpenFlow packet classification rule set [1], $N = 4$ rules, $M = 15$ fields

RID	*Ingr*	*Meta-data*	*Eth _src*	*Eth _dst*	*Eth _type*	*VID*	*Vprty*	*MPLS _lbl*	*MPLS _tfc*	*SA*	*DA*	*Prtl*	*ToS*	*SP*	*DP*
No. of bits	32	64	48	48	16	12	3	20	3	32	32	8	6	16	16
R_0	5	1024	00:13:A9:00:42:40	00:13:08:C6:54:06	0x0800	*	5	0	*	001*	*	TCP	0	*	*
R_1	*	1024	00:FF:FF:FF:FF:FF	00:13:08:C6:54:06	0x0800	100	7	163	0	00*	1011*	UDP	*	*	*
R_2	*	2048	*	00:FF:FF:FF:FF:FF	0x8100	4095	7	*	*	1*	1011*	*	*	2-1024	5-5
R_3	*	*	00:13:E6:24:5F:31	11:7B:C5:98:F0:FF	*	*	*	*	*	*	*	*	*	*	80

predefined entries for classifying a packet are called *rules*, which are stored in a *rule set*. Each rule has a rule ID (RID), multiple fields and their associated values, a priority, and an action to be taken if matched. Different fields in a rule require various types of match criteria, such as prefix match, range match, and exact match. A packet is considered matching a rule only if it matches all the fields in that rule. A packet may match multiple rules, but usually only the rule with the highest priority is used to take action.

We denote the total number of rules as N. We index all the fields as $m = 0, 1, \ldots, M - 1$, where M is the total number of packet header fields. The classic packet classification [2] requires $M = 5$ fields to be examined, while the *OpenFlow table lookup* [1] checks in total $M = 15$ fields of the packet header. In Table 1, we show an OpenFlow 15-field rule set consisting of 4 rules (omitting the actions) as an example. Our methodology in this paper can be applied to packet classification involving more than 15 fields.

2.2. Range Match

We denote the field requiring prefix match as *prefix match field*, while we define the "projection" of the rule in this field as *prefix match rule*. For example, "001*" is a prefix match rule in the *SA* field of the rule set in Table 1. Similarly, *exact match field*, *exact match rule*, *range match field*, and *range match rule* can be defined.

Many existing packet classification engines are optimized for prefix match and exact match [5], especially when TCAM is used. For range match, they usually require range-to-prefix translations [4]. This leads to *rule set expansion*: for instance, a range[1] $[1, 8)$ in a 3-bit field corresponds to a union of 3 prefixes: $\{001,\ 01*,\ 1*\}$. Note, however, any given prefix or exact value can be represented by a single (generic) range. Hence, in this paper, we propose a PE that matches ranges; this means our overall architecture can perform prefix match, exact match, and range match on any field of a packet header without rule set expansion.

3. ARCHITECTURE

To handle all types of matches, prefixes or exact values are first translated into ranges; this step is trivial so we ignore it in this paper. Suppose we have W_m-bit ranges, where $m = 0, 1, \ldots, M - 1$. A naive approach to match ranges is to deploy a W_m-bit comparator for each range boundary. However, since (1) W_m can be relatively large (*e.g.* 64 bits), and (2) the critical path in the comparator is $O(W_m)$, this naive approach often results in low clock rate.

[1]Without loss of generality, we use half-closed and half-open closures.

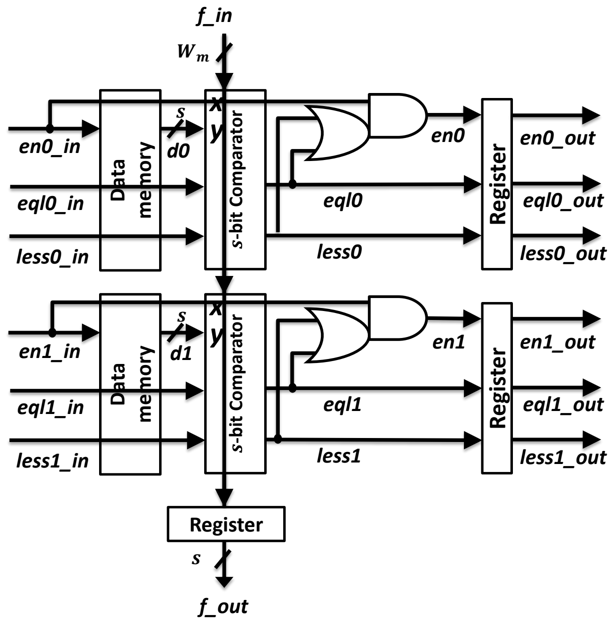


Fig. 1: A modular PE comparing an s-bit stride with $c = 2$ range boundaries in parallel

The key idea of our approach in this section is to split a range boundary in a long field into multiple shorter strides; this leads to shorter critical paths and higher clock rate. Another major difference between this work and prior works [2,4,6] is that our architecture is self-aware and can be tuned for better power consumption (see Section 4).

3.1. Modular PE

We construct a modular Processing Element (PE) to compare an s-bit stride of the input packet header against n strides of n range boundaries independently. We show the modular PE in Figure 1, where clock and control signals are omitted for simplicity. We denote the number of range boundaries compared by a modular PE as c. The modular PE in Figure 1 compares an s-bit stride of the packet header

Table 2: Truth table for a "$\leq$" comparator and the corresponding enable signal ("*" denotes "don't care")

$eql0\_in$	$less0\_in$	x and y	$eql0$	$less0$	$en0$
0	0	$*$	0	0	0
0	1	$*$	0	1	$en0\_in$
1	$*$	$x = y$	1	$*$	$en0\_in$
		$x < y$	0	1	$en0\_in$
		$x > y$	0	0	0

bits (f_in) with $c = 2$ range upperbounds ($d0$ and $d1$) concurrently. In general, a modular PE can compare c range boundaries in parallel; such a modular PE consists of c data memory modules, c comparators, the logic used to generate c enable signals, and registers.

In Figure 1, an s-bit comparator has ports x and y. The x port is fed by an s-bit stride of the input packet header; the y port is fed by the upperbound stored in the data memory. A comparator also takes partial matching results from a previous PE (*e.g.*, $eql0\_in$ and $less0\_in$) as inputs. The function of this comparator is described in Table 2.

The enable signals (*e.g.*, $en0$ and $en1$) are used to enable the data memory modules in the next modular PE; they are generated based on Table 2. Since these enable signals are very useful for power optimization, we defer the discussion of this type of signals to Section 4. The registers are used to buffer data in two dimensions:

1. *Horizontal / row-wise*: The enable signals and partial matching results are propagated in this direction.
2. *Vertical / column-wise*: The input packet headers are buffered along this direction.

3.2. Matching Ranges

For n upperbounds in a W_m-bit field ($m = 0, 1, \dots, M-1$), a row of $\lceil \frac{W_m}{s} \rceil$ modular PE can be concatenated to produce the correct partial matching result. Each PE produces the partial matching result from the highest-order stride down to the stride currently examined. Specifically:

- For the first PE in a row, the input enable signals and the inputs $eql0\_in, \dots$ are all tied to "1" (logic high); the inputs $less0\_in, \dots$ are "don't care".
- For all the modular PEs except the last one in this row, the outputs $eql0\_out$, $less0\_out, \dots$ are all connected to the corresponding inputs of the next modular PE.

For example, suppose we have an upperbound $d0 = 101101$ in a 6-bit field, and this upperbound is partitioned into 3 strides "10", "11", and "01". In this case, 3 modular PEs are required in a row. Suppose we have the corresponding field of an input packet header 100101 to be compared with this upperbound. Based on Table 2, we have:

1. The partial results generated from the first PE are $eql0 = 1$, and $less0 = *$.
2. Continuing this example, in the second PE, we have $eql0\_in = 1$, $less0\_in = *$, and $x = 01 < y = 11$; the partial results generated from the second PE are $eql0 = 0$, and $less0 = 1$.
3. The partial results generated from the second PE are propagated into the last PE. The partial results output by the last PE are $eql0 = 0$, and $less0 = 1$, indicating the input is less than the upperbound.

In summary, the correctness of our horizontal pipeline construction can be verified for all the inputs.

The modular PE shown in Figure 1 is designed for comparing the input stride with the strides of c upperbounds; a similar modular PE can be exploited for lowerbound comparisons (by switching port x and y in all the c comparators).

3.3. Systolic Array

As shown in Figure 2, there are three types of interconnections between modular PEs in the same row. The logic correctness of the interconnections can be verified easily.

Using the interconnections shown in Figure 2, we can arrange a large number of modular PEs into a systolic array, as shown in Figure 3. In this systolic array:

- The strides from the input packet headers are propagated vertically. Each column of PEs compares an s-bit input stride with the corresponding strides of all the N upperbounds/lowerbounds.
- The partial searching results, along with the enable signals, are propagated horizontally. Each row of PEs examines the input packet header against c rules.

In this example, field 0 and field 1 can be prefix/exact/range match fields. The entire design is parameterized so s and c are adjusted during design time in Section 5. In particular, to scale up the design for wider packet headers, we can add more columns of PEs; to scale up the number of rules, we can deploy more rows of PEs in the systolic array. For fixed s and c, the total number of rows required for N rules is $\lceil \frac{N}{c} \rceil$, while the total number of columns required for $\left(\sum W_m\right)$-bit packet headers is $2 \cdot \sum \lceil \frac{W_m}{s} \rceil$.

4. POWER OPTIMIZATIONS

4.1. Motivation

As can be seen in Figure 1, there are a group of enable signals propagated horizontally; each enable signal is used to enable the corresponding data memory module in the next PE. The intuitions of using the enable signals are: (1) If any field of the input packet header does not match a rule, then the entire packet header is considered as not matching this rule. (2) If a packet header has been identified as not matching the rule in one field, then the PEs in other fields can clock-gate their data memory modules to save power.

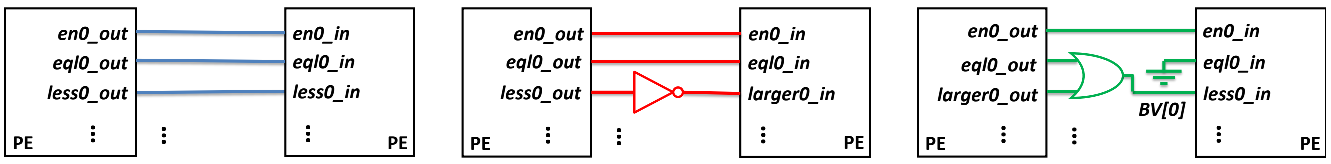


Fig. 2: Interconnections between modular PEs (from left to right): (Type I) for the same field, both comparing upperbounds/lowerbounds; (Type II) for the same field, one comparing upperbound but the other comparing lowerbound; (Type III) for different fields.

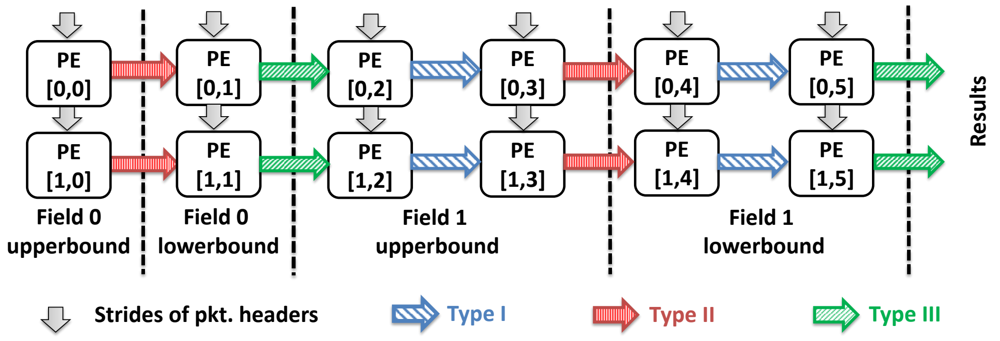


Fig. 3: Systolic array consisting of 2 rows and 6 columns, and matching $M = 2$ fields. PE$[i, j]$ denotes the modular PE in the i-th row and j-th column, where $i = 0, 1, \ldots, \lceil \frac{N}{c} \rceil - 1$, $j = 0, 1, \ldots, 2 \cdot \left(\sum \lceil \frac{W_m}{s} \rceil \right) - 1$.

If a data memory module is gated off from clock, we denote this data memory module as "*deactivated*". On the other hand, we denote the status where a data memory module is accessed regularly as "*activated*".

4.2. Self-enabled Power Gating

For the first modular PE in a specific row, its input enable signals are tied logic high. For all the other PEs in the same row, if any PE signifies a mismatch between the input packet header and the range specified by a rule, then all the PEs following this PE can have their corresponding data memory modules deactivated. Our power gating technique relies on the hardware architecture to realize the correct conditions to save power; thus, we denote this technique as *self-enabled power gating*. This technique has two properties:

1. (*Chaining*) If one data memory module is deactivated, a chain of data memory modules in the following PEs will be deactivated, saving a great amount of power.
2. (*Fine-grained*) Since the rules are independent, the activation / deactivation for different data memory modules is also independent.

For the chaining property, suppose, for example, we have an upperbound $d0 = 101101$ split into 3 strides "10", "11", and "01". Suppose we have the input 111101 to be compared with this upperbound. Based on Table 2, the first modular PE generates $eql0 = 0$, $less0 = 0$, and $en0 = 0$. This enable signal will deactivate the data memory of the second PE (storing the stride "11"); as a consequence, the data memory modules of the last PE (storing the stride "01") will also be deactivated.

For fine granularity, continuing the above example, further suppose we have another upperbound $d1 = 111111$ to be compared. Although many data memory modules storing the strides of $d0$ are deactivated, all the data memory modules storing the strides of $d1$ are activated for this input.

4.3. Entropy-based Scheduling

4.3.1. Problem Definition

To make the self-enabled power gating more effective, for a given packet header and a given rule, we need to report the mismatch, if any, as early as possible; this leads to deactivation of a large number of data memory modules in the same row due to the chaining property as discussed in Section 4.2.

We show an example in Figure 4, where 4 modular PEs are placed in a row for 2 exact match fields. For simplicity, here we assume $n = 1$, *i.e.*, a row of PEs only compare the packet headers with one rule. The first two modular PEs examine a given packet header field m, while the last two modular PEs examine another packet header field m', where $m, m' \in \{0, 1, \ldots, M - 1\}$. Consider the following two cases, as shown in this figure:

- Case (1): The input packet header matches the rule in field m, but does not match this rule in field m'. The mismatch is identified in the higher order stride of field m'. This is quite late; the 3 data memory modules in the first 3 PEs are activated.

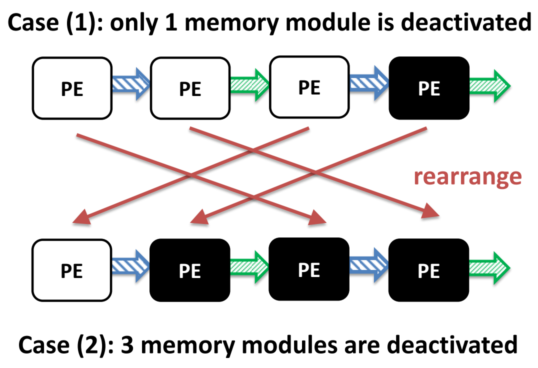

Case (1): only 1 memory module is deactivated
PE
PE
PE
PE
rearrange
PE
PE
PE
PE
Case (2): 3 memory modules are deactivated

Fig. 4: Scheduling 4 PEs. A white box denotes a PE whose data memory module is activated; a black box denotes a PE whose data memory modules is deactivated.

- Case (2): The input packet header does not match the rule in field m'; the mismatch is identified in the higher order stride of field m'. This is early enough to have 3 data memory modules deactivated for this input packet header.

Although the classification results for the above two cases are the same, the second case leads to more deactivated data memory modules and more power savings in turn.

We define a problem: *Given a rule set consisting of N rules and M fields, and a systolic array with self-enabled power gating, find an optimal (static) scheduling of all the fields to save the maximum amount of power.*

An optimal solution to this problem is difficult, since the actual power saving also depends on the input packet trace. In this paper, we assume the input values in each packet header field are uniformly distributed.

4.3.2. Greedy Algorithm

We denote the number of bits in field m as W_m, where $m = 0, 1, \ldots, M-1$. We index the rules as n, where $n = 0, 1, \ldots, N-1$. A range in field m may cover multiple values in $[0, 2^{W_m})$. Given a rule set, we denote the number of occurrences for value $k^{(m)}$ in field m as $f_{k^{(m)}}$, where $k^{(m)} \in [0, 2^{W_m})$. Therefore, the probability that an input value $k^{(m)}$ matches some rules in field m is

$$p_{k^{(m)}} = \frac{f_{k^{(m)}}}{\sum\limits_{k^{(m)}} f_{k^{(m)}}} \tag{1}$$

We show an example in Table 3, where the *entropy* of field m is given by the following equation:

$$\mathbf{H}(m) = -\sum_{k^{(m)}} \left[p_{k^{(m)}} \cdot \log p_{k^{(m)}} \right] \tag{2}$$

Based on the above notations, we propose a *greedy* algorithm based on the entropy of the packet header fields:

Table 3: An example of entropy calculation for a 2-bit field m ($W_m = 2$); the ranges specified by the rules are $[1,2)$,$[0,2)$,$[0,3)$, and $[2,4)$. The entropy is 1.91 bits.

Possible value $k^{(m)}$	0	1	2	3
No. of occurrences $f_{k^{(m)}}$	2	3	2	1
Probability $p_{k^{(m)}}$	$\frac{1}{4}$	$\frac{3}{8}$	$\frac{1}{4}$	$\frac{1}{8}$

Step 1 Calculate $\mathbf{H}(m)$, $\forall m$ using Equation 2.
Step 2 Schedule all the fields in descending order of $\mathbf{H}(m)$; *i.e.*, the fields with higher $\mathbf{H}(m)$ are always scheduled (early) in the front of the horizontal pipelines.

This algorithm can be a suboptimal solution, but it reduces the design complexity. This algorithm is intuitive because a field with higher entropy reveals more information; thus, such a field should be matched early to identify mismatches.

5. PERFORMANCE EVALUATION

5.1. Experimental Setup

We conducted experiments on the state-of-the-art Xilinx Virtex 7 FPGA (XC7VX1140T-FLG1930 -2L) [3]. This FPGA has 218800 logic slices, 1100 I/O pins, and 68 Mb BRAM; it can be configured to realize a large amount of distributed RAM (distRAM, upto 18 Mb). We evaluated the performance using Xilinx Vivado 2014.2 design tool [7]. We used the following performance metrics:

- *Throughput*: the number of packets classified per unit time (in MPPS).
- *Resource utilization*: the utilization of basic FPGA resources including logic slices, BRAM, and I/O pins.
- *Power*: the power consumption of the entire design on FPGA (in Watts).

We set a conservative clock constraint of 250 MHz for all of our designs. We reported resource utilization and clock rate using post-place-and-route reports.

We instantiated data memory modules using distRAM[2]; this design choice localized memory accesses. We exploited single-port distRAM to support self-enabled power gating. For power estimation, we fixed the temperature at 25 °C; we used Switching Activity Interchange Format (SAIF) files as inputs to Vivado power analysis tool.

Due to the lack of large real-life rule sets, especially for packet classification involving more than 5 fields [8], we built synthetic rule sets where the entropy of each field could be adjusted separately. In our synthetic rule sets, the entropies of fields *SA*, *DA*, *SP*, *DP*, and *Prtl* followed the real-life 5-tuple rule sets [9]; for other fields, the entropy of field m were assumed to be $\min\{\log N, \log\left(2^{W_m}\right)\}$ pessimistically. For fair comparisons on power, we set at least

[2]Hence the throughput is numerically equal to the clock rate.

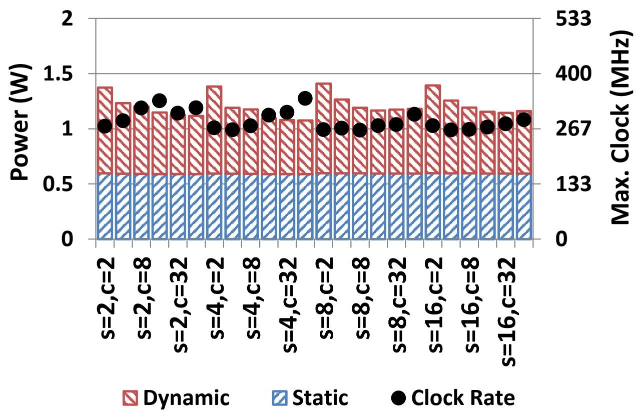


Fig. 5: Static power, dynamic power, and maximum achievable clock rate with respect to s and c

Table 4: Used I/O pins with respect to the number of rules N, each rule having $\sum W_m = 512$ bits, $s = 4$, $c = 64$

N	64	128	192	256	320	384	448	512
I/O	144	272	400	528	656	784	912	1040
%	13%	25%	36%	48%	60%	71%	83%	95%

one *wildcard* rule[3] in each rule set. We fed our packet classification engine with random packet headers.

5.2. Deciding s and c

Since our packet classification engine is fully parameterized, we first determine the values of s and c as discussed in Section 3.1. Initially, we fix the packet header length $\sum W_m = 64$; we fix the number of rules $N = 128$. For power consumption, we employ the optimization techniques in Section 4. We vary $s = 2, 4, 8, 16$, and $c = 2, 4, 8, 16, 32, 64$; we have seen similar performance results for other combinations of these parameters. In Figure 5, we show the power breakdown of our entire design at 250 MHz, and the maximum achievable clock rate on FPGA. Considering both power and clock rate, we choose $s = 4$ and $c = 64$ for our designs on FPGA. The performance of $s = 4$ and $c = 64$ is best because (1) on our FPGA, each logic slice contains 4 6-input LUTs; (2) the design with $s = 4$ and $c = 64$ uses the LUT-based distRAM as data memory modules efficiently.

5.3. Throughput

We show the throughput performance with respect to $\sum W_m$ and N in Figure 6, where $s = 4$ and $c = 64$. As can be seen, our designs on FPGA sustains > 250 MPPS throughput even for 4 K 512-bit rules. The throughput tapers as we scale up the design with respect to $\sum W_m$ or N. As the design gets larger, more resources have to be used (see Section 5.4), and the routing becomes more complex; it is more difficult for the design tool to optimize long critical paths.

[3]*i.e.*, a default action will be performed if there is no other match.

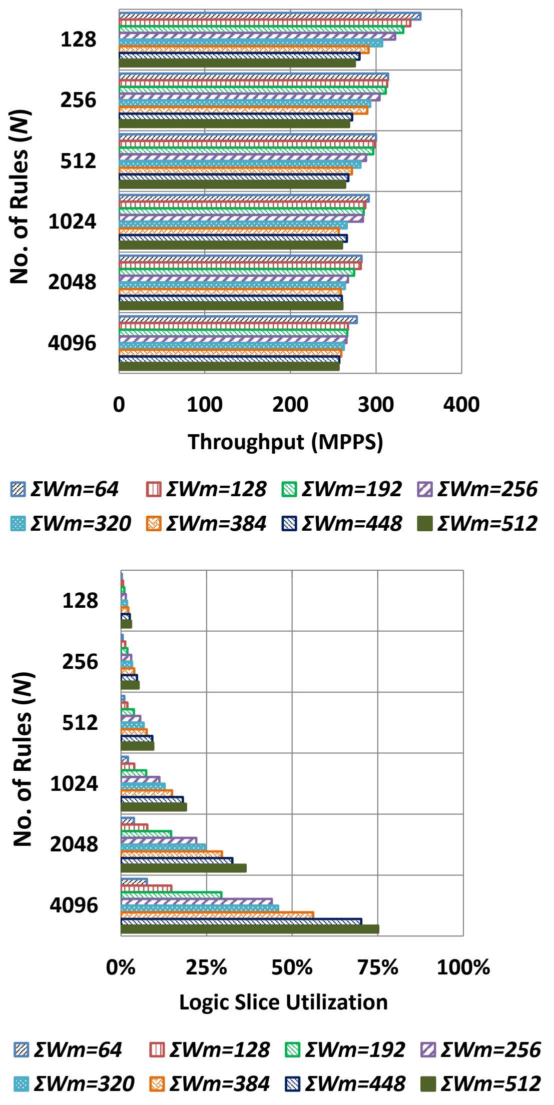
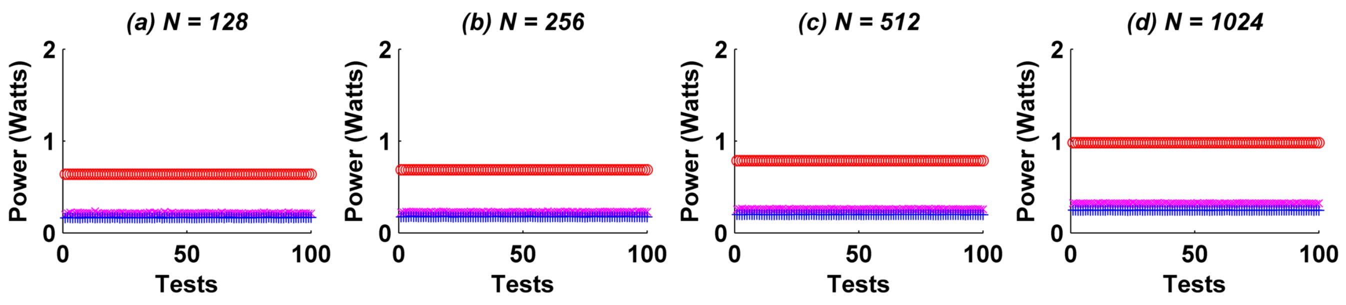


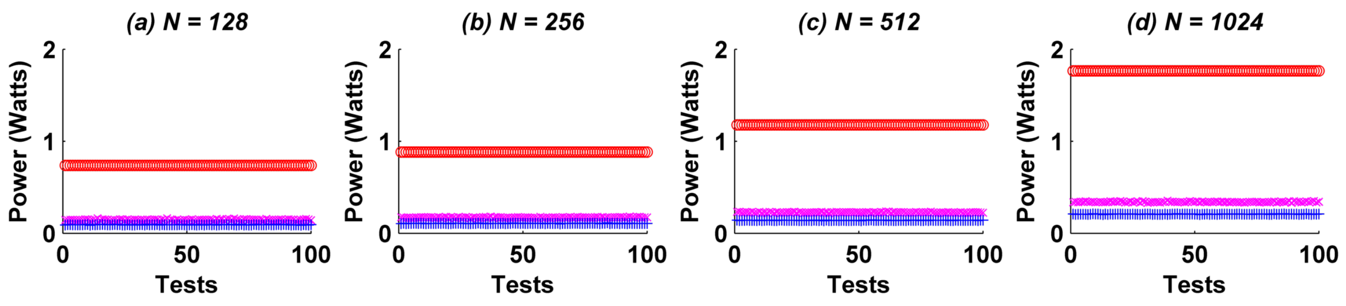
Fig. 6: Throughput and logic slice utilization with respect to the length of the packet header ($\sum W_m$) and the number of rules (N), $s = 4$, $c = 64$

5.4. Resource Utilization

Using $s = 4$ and $c = 64$, we show the utilization of logic slices with respect to $\sum W_m$ and N in Figure 6. We also show the utilization of I/O pins in Table 4. As can be seen: (1) The resource utilization on FPGA scales linearly with respect to both $\sum W_m$ and N. (2) The largest design supported on our FPGA is $\sum W_m = 512$ and $N = 4$ K, limited by the available I/O pins and logic slices on FPGA.



(i) Classic 5-field packet classification, $M = 5$, $\sum W_m = 104$



(ii) OpenFlow 15-field table lookup, $M = 15$, $\sum W_m = 356$

Fig. 7: Power consumption of 100 tests (each test processing 1 K packet headers) for $N = 128, 256, 512, 1024$, and (i) $M = 5$, $\sum W_m = 104$, and (ii) $M = 15$, $\sum W_m = 356$. A red "$\circ$" denotes the scenario where no optimization is applied. A pink "$\times$" denotes the scenario where only self-enabled power gating is used. A blue "$+$" denotes the scenario where both self-enabled power-gating and entropy-based scheduling are used; this scenario demonstrates the least power consumption.

5.5. Power Consumption

In Figure 7, we show the power performance with respect to N, for the classic packet classification $(M = 5)$ and the OpenFlow table lookup $(M = 15)$ under three scenarios:

1. Without any power optimization technique.
2. With self-enabled power gating, but randomly scheduling of packet header fields.
3. With self-enabled power gating along with entropy-based scheduling of packet header fields.

To see the effect of random scheduling, we run 100 tests for a fixed pair of M and N, each test using a specific (possibly different) scheduling of packet header fields. For each test, the power consumption is averaged on 1 K random packets. As can be seen in Figure 7:

- Compared to the designs without any power optimization techniques, the designs with self-enabled power gating reduce the average power consumption by 60%.
- By also exploiting entropy-based scheduling of packet header fields, our designs reduce the average power consumption further by 20%.
- As the packet header gets longer (by increasing M), more and more fields are gated away from matching the input packet header; therefore we observe even more power savings for larger M.

Using our power optimization techniques, the average power consumption of our largest design ($\sum W_m = 512$ and $N =$ 4 K) is 1.655 Watts; this is only 30% of the power consumed by a design without any optimization, 5.674 Watts.

6. RELATED WORK

6.1. Decision-tree vs. Decomposition

There are two major categories for packet classification approaches [2]: *decision-tree-based* and *decomposition-based* approaches. The decision-tree-based approaches employ several heuristics to cut the space recursively into smaller subspaces [4, 13]. The major advantage of a decision-tree-based approach is that various optimization techniques can be applied [8]. However, the performance of a decision-tree-based approach depends on the statistics of the rule set. Also, for an M-field rule set consisting of N rules, the recursive cutting can consume $O(N^M)$ memory; this is very expensive.

In decomposition-based approaches, the fields of an input packet header are searched independently; the partial results of all the fields are merged to produce the final matching result [14, 15]. The main advantage of the decomposition-based approaches is that parallel data structures can be explored. However, it can take either $O(N^M)$ memory or $O(MN)$ time to merge all the partial results from M fields.

Table 5: Performance comparison

	Approach	Platform	Match Type	Rule Sets		Throughput	Throughput $\times N \times M$
				No. of rules, N	No. of fields, M	(MPPS)	(MPPS$\times$1 K)
Ma *et al.* [10]	decision-tree	multi-core GPP	any	9 K	5	$\sim$ 14	$\sim$ 630
Han *et al.* [5]	decomposition	GPP + GPU	prefix / exact	32 K	10	$\sim$ 58	$\sim$ 18560
Qu *et al.* [11]	decomposition	multi-core GPP	any	32 K	15	$\sim$ 30	$\sim$ 14400
Song *et al.* [12]	decomposition	FPGA + TCAM	any	1.2 K	5	$\sim$ 20	$\sim$ 120
Kennedy *et al.* [13]	decision-tree	FPGA	any	60 K	5	$\sim$ 15	$\sim$ 4500
Pus *et al.* [14]	decomposition	FPGA	prefix / exact	0.6 K	5	$\sim$ 500	$\sim$ 1500
Jiang *et al.* [8]	decision-tree	FPGA	any	5 K	12	$\sim$ 125	$\sim$ 7500
This work	decomposition	FPGA	any	4 K	15	$>$ 250	$>$ 15000

6.2. Platforms

Excluding TCAM, algorithmic solutions for packet classification mainly exploit FPGA, multi-core General Purpose Processor (GPP), or/and Graphics Processing Unit (GPU) platforms. Without using off-chip memory, FPGA often supports high throughput for rule sets of moderate size [8]. With an optimized memory hierarchy, a multi-core GPP [10] or a GPU-accelerated platform [5] can employ parallel algorithms and support very large rule sets. Our designs in this paper employ a decomposition-based approach on FPGA; the resulting architecture is a systolic array supporting generic range match and efficient power optimization techniques.

6.3. Comparison

We compare our work with the existing works in Table 5. For the prior works on multi-core GPP or/and GPU, Ma *et al.* [10] exploit 2$\times$ 4-core Intel Xeon X5550 running at 2.7 GHz and assume the availability of TCAM. Han *et al.* [5] employ 2$\times$ 4-core Intel Xeon X5550 running at 2.7 GHz, and 2 NVIDIA GTX480 cards. Qu *et al.* [11] exploit 2$\times$ 8-core AMD Opteron 6278 running at 2.4 GHz. For fair comparisons, the results of the prior works on FPGA [8, 13, 14] are all scaled up on the state-of-the-art Xilinx Virtex-7 platform; this is done by considering (1) the maximum clock frequency and (2) the maximum memory capacity on these platforms. Since many designs are usually constrained by other FPGA resources (*e.g.*, limited I/O pins), the throughput estimation for the prior works on FPGA are optimistic.

We use the product of the throughput, the number of rules (N), and the number of fields (M) as a compound performance metric[4]. As can be seen in Table 5, compared to [5], our design achieves similar performance with respect to this compound performance metric; however, our design supports generic range match much more efficiently without any rule set expansion. With respect to this compound performance metric, our packet classification engine achieves at least 2$\times$ performance compared to the existing works on FPGA.

[4]The same amount of hardware resources are required in the following two cases: (1) by replicating a design α times, we can achieve $\alpha\times$ throughput for the same rule set; (2) by replicating a design α times, we can support $\alpha\times$ larger rule sets while sustaining the same throughput.

7. CONCLUSION

We presented a range-match-based packet classification engine on FPGA in this paper. We concatenated modular PEs into a systolic array, and explored efficient power optimization techniques. Many existing works have not provided power consumption results; we will compare the power consumption of our work with prior works in the future.

8. REFERENCES

[1] "OpenFlow Switch Specification V1.3.1," https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf.

[2] D. E. Taylor, "Survey and Taxonomy of Packet Classification Techniques," *ACM Computing Surveys*, vol. 37, no. 3, pp. 238–275, 2005.

[3] "Virtex-7 FPGA Family," http://www.xilinx.com/products/virtex7.

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

[5] S. Han, K. Jang, K. Park, and S. Moon, "PacketShader: A GPU-accelerated Software Router," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 195–206, 2010.

[6] Y. R. Qu, S. Zhou, and V. K. Prasanna, "High-performance Architecture for Dynamically Updatable Packet Classification on FPGA," in *Proc. ACM/IEEE ANCS*, 2013, pp. 125–136.

[7] "Vivado Design Suite," http://www.xilinx.com/products/design-tools/vivado.html.

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

[9] "Evaluation of Packet Classification Algorithms," http://www.arl.wustl.edu/~hs1/PClassEval.html.

[10] 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, pp. 227–238, 2010.

[11] Y. R. Qu, S. Zhou, and V. K. Prasanna, "Scalable Many-Field Packet Classification on Multi-core Processors," in *Proc. IEEE SBAC-PAD*, 2013, pp. 33–40.

[12] H. Song and J. W. Lockwood, "Efficient Packet Classification for Network Intrusion Detection using FPGA," in *Proc. ACM/SIGDA FPGA*, 2005, pp. 238–245.

[13] A. Kennedy, X. Wang, Z. Liu, and B. Liu, "Low Power Architecture for High Speed Packet Classification," in *Proc. ACM/IEEE ANCS*, 2008, pp. 131–140.

[14] V. Pus, J. Korenek, and J. Korenek, "Fast and Scalable Packet Classification using Perfect Hash Functions," in *Proc. ACM/SIGDA FPGA*, 2009, pp. 229–236.

[15] S. Dharmapurikar, H. Song, J. Turner, and J. Lockwood, "Fast Packet Classification using Bloom Filters," in *Proc. ACM/IEEE ANCS*, 2006, pp. 61 –70.