Applied Soft Computing 38 (2016) 933–941

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Evolutionary circuit design for fast FPGA-based classification of network application protocols☆

D. Grochol, L. Sekanina*, M. Zadnik, J. Korenek, V. Kosar

Brno University of Technology, Faculty of Information Technology, IT4Innovations Centre of Excellence, Bozetechova 2, 61266 Brno, Czech Republic

ARTICLE INFO

Article history:
Received 10 July 2015
Received in revised form 1 September 2015
Accepted 24 September 2015
Available online 28 October 2015

Keywords:
Application protocol
Classifier
Cartesian genetic programming
Field programmable gate array

ABSTRACT

The evolutionary design can produce fast and efficient implementations of digital circuits. It is shown in this paper how evolved circuits, optimized for the latency and area, can increase the throughput of a manually designed classifier of application protocols. The classifier is intended for high speed networks operating at 100 Gbps. Because a very low latency is the main design constraint, the classifier is constructed as a combinational circuit in a field programmable gate array (FPGA). The classification is performed using the first packet carrying the application payload. The improvements in latency (and area) obtained by Cartesian genetic programming are validated using a professional FPGA design tool. The quality of classification is evaluated by means of real network data. All results are compared with commonly used classifiers based on regular expressions describing application protocols.

1. Introduction

Evolutionary algorithms (EAs) are traditionally used in the circuit design community mainly as efficient optimization techniques. In recent years, significant developments and progress in evolutionary circuit design have been witnessed. In many cases these techniques were capable of delivering efficient circuit designs in terms of an on-chip area minimization (e.g. [1]), adaptation (e.g. [2]), fabrication variability compensation (e.g. [3]), and many other properties (see, for example, many requirements on synthetic benchmark circuits in [4]). In this paper, it is exploited that the evolutionary design can produce fast and efficient circuit implementations. One of the targets is the circuit latency which is a crucial parameter in high performance computing and other applications such as security monitoring of high speed computer networks or high frequency trading. The objective of this work is to minimize the latency and area of key circuits needed in a hardware accelerator intended for classification of application protocols in high speed networks. The classifier is embedded into a software defined monitoring (SDM) platform (see details in Section 2) which is accelerated in a field programmable gate array (FPGA) [5].

In order to identify the application (or the application protocol) the network traffic belongs to, one has to inspect one or several packets with a payload. The main difficulty is that the time to process one packet is less than 7 ns in the case of modern 100 Gbps link. Hence this task has to be performed by specialized hardware. In previous work of the authors [6], key circuit components were developed for an FPGA-based application protocol classifier in which the area and latency were optimized by means of Cartesian genetic programming (CGP). The resulting circuit enabled to classify three application protocols (HTTP, SMTP, SSH) using the first packet carrying the application payload. This circuit, in fact, implemented a deterministic parallel combinational signature matching algorithm in the FPGA.

A more significant latency and area reduction, which will be crucial for classifiers supporting throughputs beyond 100 Gbps, is possible either by using advanced (faster) hardware or changing the packet processing scenario. In this paper, a new approach is proposed with respect to [6] in which small errors in the hardware protocol classification are tolerated assuming that latency and area of the classifier are significantly reduced. This concept is supported by SDM because the traffic unclassified in the hardware can be sent to the software for detailed processing.

Within this scope, the proposed work focuses on a design and optimization of three proprietary circuits, operating as application protocol classifiers, which differ in the quality of classification,

☆ This paper is an extended, improved version of the paper A Fast FPGA-Based Classification of Application Protocols Optimized Using Cartesian GP presented at EvoComNet2015 and published in: Applications of Evolutionary Computing, Proceedings of 18th European Conference, EvoApplications 2015, Copenhagen, Denmark, April 8–10, 2015, LNCS 9028, pp. 67–78, Springer, 2015.

* Corresponding author. Tel.: +420 541141215.
E-mail addresses: igrochol@fit.vutbr.cz (D. Grochol), sekanina@fit.vutbr.cz (L. Sekanina), izadnik@fit.vutbr.cz (M. Zadnik), korenek@fit.vutbr.cz (J. Korenek), ikosar@fit.vutbr.cz (V. Kosar).

http://dx.doi.org/10.1016/j.asoc.2015.09.046
1568-4946/© 2015 Elsevier B.V. All rights reserved.

latency and area. Classifier CL-acc (accuracy) is implemented according to [6] with the goal to minimize the classification error. While classifier CL-cmp (compromise) provides a moderate compromise between the latency, area and classification accuracy, classifier CL-lat (latency) is highly optimized for a low latency. Each classifier is evaluated in the task of classification of four protocols (HTTP, SMTP, SSH, and SIP) we deem most crucial from the perspective of network monitoring. It should be noted that SIP has not been considered in the initial study [6].

The main contribution of this paper is to show that these circuit classifiers can be optimized by CGP in order to significantly reduce their latency and resources requirements. The classification algorithm is not optimized by CGP. The improvements in latency (and area) obtained by CGP are validated using a professional FPGA design tool. The quality of classification is evaluated by means of real network data. All results are compared with commonly used classifiers based on regular expressions describing application protocols. Contrasted to [6], in which only key components of one classifier were implemented and optimized, complete FPGA implementations of three classifiers are evaluated.

The rest of the paper is organized as follows. Section 2 briefly surveys the field of traffic analysis in high speed networks, accelerated network technologies using FPGAs and evolutionary circuit design. Section 3 provides a specification of the classifier and network data used for the evaluation. In Section 4, the proposed hardware classifier and its approximations are introduced. Cartesian genetic programming is presented as a digital circuit design and optimization method in Section 5. Section 6 describes the implementation steps taken and the results in terms of area and latency in the FPGA. Finally, the quality of classification is assessed in terms of precision and recall. Conclusions are given in Section 7.

2. Relevant work

This paper deals with several different research areas – network traffic analysis in high speed networks, FPGA technology, fast pattern matching and evolutionary circuit design. The purpose of this section is to provide an appropriate introduction to them and to their intersections which are relevant for the target application.

2.1. Traffic analysis in high speed networks

An abstract yet detailed network traffic visibility is a key prerequisite to network management, including tasks such as traffic engineering, application performance monitoring and network security monitoring. In recent years the diversity and complexity of network applications and network threats have grown significantly. This trend has rendered monitoring of network and transport layer insufficient and it has become important to extend the visibility into the application layer, primarily to identify the application (or the application protocol) the traffic belongs to. The port numbers are no longer reliable application differentiators due to new emerging applications utilizing ports dynamicaly or to applications evading the firewalls by hiding behind well-known port numbers or utilizing port numbers defined by users [7].

The research in the area of application identification has come up with distinct approaches to identify applications carried in the traffic. These approaches differ in the level of detail that is utilized in the identification method. The most abstract one is behavioral analysis [8,9]. Its idea is to observe only the port number and destination of the connections per each host and then to deduce the application running on the host by its typical connection signature. If more details per connection are available, statistical fingerprinting [10] comes into play. In this case, a feature set is collected per each flow and the assumption is that the values of the feature set vary across applications and hence they leave a unique fingerprint. Behavioral and statistical fingerprinting generally classifies traffic to application classes rather than to particular applications. The reason is that different applications performing the same task exhibit similar behavior. For instance, application protocols such as Oscar (ICQ), MSN, XMPP (Jabber) transport interactive chat communications and hence exhibit a similar behavior, which makes it very hard to differentiate between them. The inability to distinguish applications within the same class is seen as a drawback in some situations when, for example, it is necessary to block a particular application while allowing others in the same class. The approach utilizing the greatest level of detail is a deep packet inspection. It identifies applications based on the packet payload. The payload is matched with known patterns (defined, for example, by regular expressions) derived for each application [11].

The application identification poses several on-going challenges. The identification process is bound to keep pace with ever increasing link speeds (for example, the time to process each packet is less than 7 ns in the case of a 100 Gbps link). Another challenge is represented by the growing number of protocols (i.e., the application identification must address trends such as new emerging mobile applications or applications moving into the network cloud [12]). Some deployments of application identification also require prompt (near real-time) identification to enable implementation of traffic engineering or application blocking [13].

Hardware acceleration (e.g. utilizing an FPGA) is often employed to speed up network processing [14,15], including the application identification directly on the network card. An FPGA renders it possible to utilize various pattern matching algorithms to identify applications. However, pattern matching may exhibit several constraints, that is, the high cost to process wide data inputs (which is the case for high throughput buses in FPGA) and the high complexity and overhead of a pattern matching algorithm which consumes valuable hardware resources or constrains the achievable frequency.

These drawbacks are addressed by alternative methods which look for constants and fixed-length strings (for brevity they are called the *signatures* in the paper) rather than regular expressions (e.g. [16]). This paper builds upon this strategy and envisions a hardware-software codesign approach in which a simple circuit labels the traffic belonging to applications of interest with some probability of false positives while software can subsequently handle and check the labeled traffic with a more complex algorithm effectively. This approach is supported by the software defined monitoring concept [5]. Software defined monitoring employs sophisticated processes running in the software to subsequently install rules in the hardware (network card). While it is not possible (or at a very high cost) to process all traffic in the software, the application identification is offloaded into the hardware. The offload not only reduces the host memory and processor load but it also increases the expressive strength of the SDM rules. The target applications range from application-specific forwarding and traffic shaping to traffic monitoring and blocking.

2.2. FPGAs in network applications

Performance requirements are growing due to the increasing volume and rates of network traffic. Paxson et al. [17] argue that these performance requirements should be met by leveraging a high degree of possible parallelism that is inherent to network traffic monitoring. FPGAs as well as ASICs may deliver such a vast support of parallelism. However, only FPGAs render it possible to prototype and implement critical application components for various network applications at the highest speeds while the optimized ASICs follow after broad deployment a few years later on. FPGAs are thus extensively used in the so-called hardware-accelerated network cards to implement the first line of network traffic processing, such as monitoring, forwarding and other applications [18,19].

FPGAs include a high spectrum of components, but the following components are crucial for the purposes of this paper. FPGAs consist of routing network and basic building blocks such as look-up tables (LUTs), registers and block memories. The particular setup of the routing network defines the interconnection of these components (i.e. the layout of the circuit). The LUTs serve to implement combinational logic while registers and block memories serve to keep the stateful information. Modern FPGAs contain millions of LUTs and registers and thousands of block memories. All these components may, in theory, work in parallel independet of each other providing enormous computation power with a low energy consumption in tens of Watts. Moreover, FPGAs targeting the network market include more than a hundred of high-speed transceivers allowing for connection to high speed network links (e.g. high-end Virtex UltraScale+ FPGA offers up to 4Tbps of aggregated transceiver throughput [20]). The crucial task is to transform a high-level description of the circuit (for example, written in VHDL or SystemC) into an effective implementation in FPGA from the perspective of meeting the timing and resource constraints.

2.3. Fast pattern matching

The L7 filter [21] is a popular program for application protocol identification, which utilizes regular expressions to describe application protocols. It performs pattern matching in network flows. If a known pattern is matched in the payload, the corresponding application protocol is assigned to the network flow. Current processors are not powerful enough to achieve 100 Gbps throughput for regular expression matching. The throughput of L7 decoder is less than 1 Gbps per one CPU core even for the latest Xeon processors. In order to achieve 100 Gbps throughput, it is necessary to use highly optimized hardware architectures.

In recent years, many researchers have proposed high-speed pattern matching hardware architectures, which utilize the fine grained parallelism of FPGA technology. Mapping of regular expressions matching to an FPGA was first explored by Floyd and Ullman [22], who showed that a Nondeterministic Finite Automaton (NFA) can be implemented using a programmable logic array. Sindhu et al. [23] proposed efficient mapping of NFAs to FPGA and Clark et al. improved the mapping by a shared decoder [24,25] which significantly reduced the amount of consumed logic resources. The AMTH (At Most Two-Hot encoding) [26] architecture improves NFA mapping to the FPGA. The combination of one-hot and binary encoding reduces the amount flip-flops, which represent NFA states.

Several papers introduced optimized mapping of Perl Compatible Regular Expressions (PCRE), which are widely used in Intrusion Detection Systems (IDS). Sourdis et al. published in [27] an architecture that allows for the sharing of character classes, static subpatterns and introduced components for efficient mapping of constrained repetitions to the FPGA. Lin et al. created an architecture for sharing infixes and suffixes [28]. Nevertheless, these optimizations are relevant only for large sets of PCRE in IDS systems. In this work, a small set of regular expressions without counting constraints and other advance PCRE constructions is only used. Therefore, these optimizations are not considered in the evaluation of proposed architectures.

The throughput of pattern matching is determined by the amount of bytes processed within one clock cycle and frequency of the hardware matching unit. The FPGA technology limits the maximum frequency to several hundreds of MHz. To increase the processing speed, the NFA can be modified to process multiple bytes per one clock cycle [29]. Unfortunately, with the increasing size of the NFA input, the amount of NFA transitions grows exponentially.

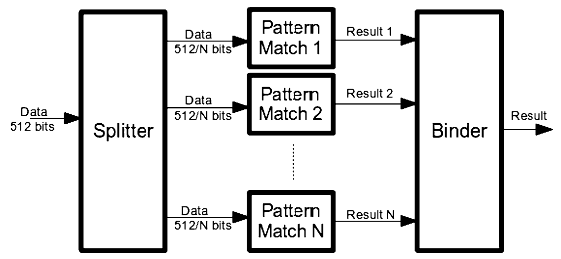

Fig. 1. Increasing the throughput by multiple pattern matching units.

As a result, the hardware matching unit consumes much more FPGA resources and the frequency decreases rapidly.

The throughput can be increased by multiple parallel matching units. These units need additional logic resources and buffers to distribute network data to the matching units and join the results. The overhead of parallel processing is shown in Fig. 1. First, the splitter has to assign the sequence number into every packet and store the packet to the buffer. The packet data are then sent with a lower rate to parallel matching units. The units perform pattern matching and provide the results to the binder, which needs buffers to order the results in the right sequence order.

It can be seen that the parallel matching units can scale the matching speed up to 100 Gbps throughput, but only at the cost of significant overhead in terms of latency, FPGA logic resources and memory buffers. This overhead is avoided by focusing on highly optimised hardware architectures with high throughput and low latency.

2.4. Evolutionary circuit design

The idea of evolvable hardware and automated circuit design by means of artificial evolution was introduced by Higuchi et al. in 1993 [30]. A recent survey of the field covering key subfields (evolutionary hardware design and adaptive hardware) is available in [31]. Significant progress in the evolution of digital circuits is connected with Cartesian genetic programming which has been developed by Miller since 1999 and utilized in many applications as documented in the recent monograph [32]. Since only combinational circuits will be evolved in this work, CGP is a natural choice.

CGP is a form of genetic programming in which candidate designs are represented using directed oriented graphs (see a detailed description in Section 5). In the standard CGP used for combinational circuit evolution, each candidate circuit is directly mapped into a chromosome consisting of a string of integers and evaluated by applying all possible input vectors. Although various new designs have been discovered using CGP, the method is not directly applicable for the design of large combinational circuits because the fitness evaluation time grows exponentially with the number of primary inputs. Moreover, the number of evaluations can easily go into the millions, even for small (but nontrivial) circuits such as multipliers. This problem has partially been eliminated by introducing circuit decomposition techniques at the representation level [33,34] and formal verification methods in the fitness function [1]. Other successful applications of CGP have been proposed in domains in which candidate circuits are not evaluated using all possible input combinations (see e.g. hash functions [35], image operators [36] or classifiers [2]).

In order to accelerate the fitness function evaluation on a common processor, a bit-level parallel simulation of candidate combinational circuits is employed. Contrasted to a naïve simulation, in which 2^k vectors are sequentially submitted for evaluation (where k is the number of primary inputs), the bit-level parallel simulation exploits the fact that current processors enable performing bitwise operations over two w-bit operands in parallel.

Table 1
The flows corresponding to the application protocols in data sets.

Data set	CESACO		CESPIO		DATASET SIP	
Protocol	Count flows	Count flows [%]	Count flows	Count flows [%]	Count flows	Count flows [%]
HTTP	1914	38.12	15060	52.29	134	2.41
SMTP	4	0.08	34	0.12	10	0.18
SSH	1	0.02	0	0.00	14	0.25
SIP	0	0	0	0	5204	93.42
Others	3102	61.78	13705	47.59	208	3.74
All	5021	100.00	28799	100.00	5570	100.00

Hence the input vectors are grouped into *w*-bit words and simulated in parallel. The obtained speedup is *w* on a *w*-bit processor, for example, 64 on a common personal computer. Even if this approach is taken, a typical CGP run could take tens of minutes for a circuit with 8 inputs and 8 outputs.

There are only a few papers dealing with evolutionary circuit design at the level of 4-input LUTs [35,37] and no paper dealing with 6-input LUTs. Unfortunately, the bit-level parallel simulation is inefficient for circuits consisting of LUTs because their logic function has to be emulated using a sequence of binary logic operations. Moreover, employing CGP with 6-input LUTs (each of them encoded using 64 bits in the chromosome) would lead to long chromosomes, complex search spaces and very inefficient search procedures. Hence two-input gates represent the dominant option when CGP is applied to the evolution of complex circuits.

3. Requirements and network data

In order to design, implement and evaluate an FPGA-based application protocol classifier, its basic parameters and an environment in which it will be operated have to be specified.

3.1. Specification of the classifier

The classifier has to distinguish among four application protocols (HTTP, SMTP, SSH and SIP) which represent an important portion of the network traffic and play an important role in traffic monitoring. Remaining protocols will be classified as unknown. Because the primary goal is achieving a very low latency, only signatures of the first packet carrying the application payload will be defined and utilized in the classifier architecture. The classifier will operate in an FPGA on a 512 bit bus to meet the 100 Gbps throughput. The application payload may start at nearly arbitrary offset (byte of a word) on the bus and the application (protocol) must be identified each clock cycle to keep pace even with the shortest incoming packets of 64 bytes.

The classifier will be constructed manually – as a combinational circuit with a low latency. CGP will be applied to optimize its key subcircuits to reduce the latency and area. An observation is utilized that a circuit which is well optimized by a commercial FPGA synthesis tool can further be re-synthesized and re-optimized by CGP to improve its parameters (see example circuits created by this approach in [36]). Such a classifier will be considered as a fully functional solution (CL-acc).

Further area and latency improvement are obtained if the requirement of full functionality can be relaxed. Hence we will also propose and evaluate classifiers (CL-cmp and CL-lat) showing a shorter latency and smaller area. Providing such approximations is currently a hot topic in computer engineering. The approach is called *approximate computing* and its goal is to investigate how computer systems can be made better – more energy efficient, faster, and less complex by relaxing the requirement that they are exactly correct [38].

3.2. Network data

The data which has to be classified are common network data (available in the pcap format). In our case, complete network data sets with anonymized IP addresses are utilized, collected on CESACO link (connecting CESNET and ACONET networks) and CESPIO link (connecting CESNET and PIONIER networks), see Table 1. Because SIP and SSH are not adequately present in these data sets, another, dedicated data set (DATASET SIP) with a high presence of SIP records was employed.

For example, the available record from CESPIO contains 43 M packets, where percentages are 78.72% for TCP, 20.58% for UDP, 0.18% for ICMP and 0.53% others. One can observe that only TCP and UDP are relevant for our purposes. The packet traces were analyzed using Scapy. In the case of HTTP, SMTP and SSH, which operate over TCP, the third or the fourth packet of the TCP connection is usually considered as the first packet containing the application payload. The L7 filter [21] was utilized as a reference classifier to annotate each connection in the data set.

The resulting data sets, which can be used for evaluation purposes, are available in the JSON format. Each record contains the source IP and port, the destination IP and port, the transport protocol number, and the whole packet encoded using base64 (see Fig. 2). Table 1 gives the mix of considered protocols in our data sets.

4. Proposed classifiers

This section describes the analytical approach taken in order to construct the proposed classifiers. Detailed hardware architecture of the classifiers is then presented.

4.1. Deterministic classification

Because the classification utilizes only the start of the payload, several initial bytes of considered application protocols were analyzed and characters were identified which are unique in these protocols. Table 2 shows the unique *signatures* that were identified for considered protocols. The longest signature of the CL-acc contains 10 characters (bytes). Signatures of classifier CL-cmp are constructed from those used in CL-acc in such a way that they are reduced to the first 4 characters, which leads to less complex hardware. Further area and latency reduction is expected in classifier CL-lat which operates with signatures containing at least 3 characters, but each of them has to exist in at least two signatures of CL-acc.

```
{
 ”dIP”: ”192.168.0.2”,
 ”dPort”: 80,
 ”data”: ”R0VUIC9zaXRlcy9kZWZhdWx0L3RoZW1lcy9mcmFtZWR5bmFtaWMv...
 ”id”: ”(’ 192.168.0.1’, ’192.168.0.2’, 52217, 80)”,
 ”trProto”: 6,
 ”protocol”: ”HTTP”,
 ”sIP”: ” 192.168.0.1”,
 ”sPort”: 52217
},
```

Fig. 2. Example of record in the data set.

Table 2
Unique signatures in considered application protocols.

Protocol	CL-acc	CL-cmp	CL-lat
HTTP	“GET /”	“GET ”	“*ET /”
	“PUT /”	“PUT ”	“*UT /”
	“POST /”	“POST”	“*OS* /”
	“HEAD /”	“HEAD”	“*EA* /”
	“TRACE /”	“TRAC”	“T*ACE**”
	“DELETE /”	“DELE”	“*E**TE**”
	“OPTIONS /”	“OPTI”	“**TI*NS**”
SIP	“INVITE ”	“INVI”	“*N*ITE”
	“REGISTER ”	“REGI”	“*E*IS*E*”
	“CANCEL ”	“CANC”	“C**CE*”
	“MESSAGE ”	“MESS”	“*ESS**E”
	“SUBSCRIBE ”	“SUBS”	“SU*SC***E*”
	“NOTIFY ”	“NOTI”	“*OTI**”
SSH	“SSH-”	“SSH-”	“SS*-”
SMTP	“220 ”	“220 ”	“220 ”
	“220-”	“220-”	“220-”

These classifiers can be constructed as combinational circuits by means of a decoder. However, they have to correctly manage the cases in which the signatures appear at various offsets within the frame due to preceding protocol headers, which is a natural situation in real network traffic data.

4.2. Classifiers in hardware

The hardware architecture utilizes a 512 bit bus to transfer protocol frames. Each frame starts with the headers of low-level protocols such as Ethernet, IPv4 or IPv6, TCP or UDP. As a result, the start of the application payload may appear with certain offsets on the bus, namely 2 bytes from the position 0 or with $2 + 4k$ bytes, where $k = 1, \ldots, 16$.

All three versions of the classifier are constructed according to Fig. 3 which also shows that the circuit classifier consists of three levels of combinational logic.

Table 3
CL-acc: mapping functions in the coders. The * symbol means: “not utilized in a particular coder”. ω stands for “otherwise”.

Coder 1	Coder 2	Coder 3	Coder 4	Output
Space	Space	Space	Space	00000011
/	/	/	/	00000101
2	2	0	–	00000110
A	A	A	B	00001001
C	E	B	C	00001010
D	G	E	D	00001100
E	L	G	E	00010001
F	N	H	I	00010010
G	O	I	R	00010100
H	P	L	S	00011000
I	R	N	T	00100001
M	S	S	*	00100010
N	T	T	*	00100100
O	U	V	*	00101000
P	Y	*	*	00110000
R	*	*	*	01000001
S	*	*	*	01000010
T	*	*	*	01000100
ω	ω	ω	ω	00000000

In the first level, one coder is connected to each byte of the word (64 coders, in total). There are four types of the coders (c1, c2, c3, c4) because of the 4-byte offsets. Each coder implements a mapping from the set of characters allowed for the given position to a set of 8-bit values in which just 2 bits are not zeros. The mapping functions of the coders in CL-acc, CL-cmp and CL-lat are given in Tables 3–5.

This remapping implemented by coders allows for a fast signature detection in the subsequent level of comparators. All possible occurrences of the application data within the input word are thus processed in parallel.

The second level consists of comparators. In the case of CL-acc, each of them compares the outputs of ten coders (note that the longest signature contains 10 characters) with the unique patterns identified for the considered application protocols. If a particular application protocol is detected then its 4-bit code is visible at the output of the comparators (0001 – HTTP, 0010 – SMTP, 0100 – SSH,

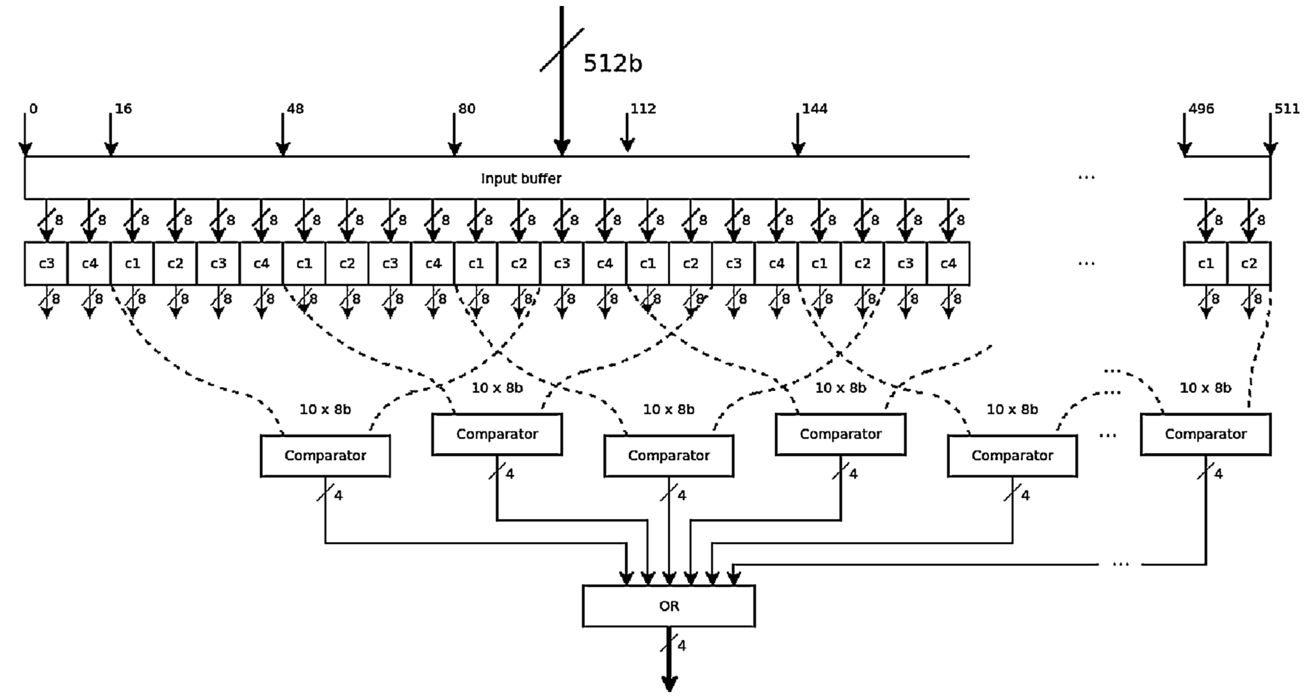

Fig. 3. Classifier CL-acc as a combinational circuit.

Table 4
CL-cmp: mapping functions in the coders.

Coder 1	Coder 2	Coder 3	Coder 4	Output
2	2	0	Space	00000011
C	A	A	–	00000101
D	E	B	C	00000110
G	N	G	D	00001001
H	O	H	E	00001010
I	P	L	I	00001100
M	R	N	S	00010001
N	S	S	T	00010010
O	U	T	*	00010100
P	*	V	*	00011000
R	*	*	*	00100001
S	*	*	*	00100010
T	*	*	*	00100100
ω	ω	ω	ω	00000000

Table 5
CL-lat: mapping functions in the coders.

Coder 1	Coder 2	Coder 3	Coder 4	Output
Space	/	Space	Space	00000011
/	2	0	–	00000101
2	E	A	C	00000110
C	N	E	I	00001001
E	S	S	S	00001010
S	O	T	*	00001100
T	U	*	*	00010001
ω	ω	ω	ω	00000000

1000 – SIP, 0000 – unknown). In the case of CL-cmp (CL-lat, respectively) the circuit is simplified as only 4 (9, respectively) coders are employed. Finally, at the third level, all 4-bit codes are fed to an OR gate which indicates a presence of the detected application protocols or unknown protocol (0000).

5. Coder evolution using CGP

Based on our previous experience, it is assumed that parameters of a circuit optimized by a professional FPGA design software can be improved if CGP is employed [36]. As the whole classifier is a relatively complex circuit to be optimized, it is proposed to evolve its components – 64 (combinational) coders. Each of the coder types c1, c2, c3 and c4 will be evolved by CGP separately. The standard CGP is used as defined in [32].

In CGP, a candidate circuit is modeled as a directed acyclic graph and represented in a 2D array of $n_c \times n_r$ processing nodes. Each node is capable of performing one of the n_a-input functions specified in Γ set. The setting of n_c, n_r and Γ significantly influences the performance of CGP [39,40]. Current FPGAs utilize 6-input LUTs as building blocks of all circuits. However, employing CGP with 6-input nodes (each of them encoded using $2^6 = 64$ bits in the chromosome) would lead to long chromosomes, complex search spaces and so inefficient search procedures. It is proposed to optimize the coders at the level of 2-input nodes (encoded using up to 4 bits) and let the professional circuit synthesis software implement the resulting optimized circuits using 6-input LUTs in the FPGA.

The remaining parameters of CGP are the number of primary inputs (n_i), the number of primary outputs (n_o), and the level-back parameter (L) specifying which nodes can be used as inputs for a given gate. The primary inputs and the outputs of nodes are labeled $0 \ldots n_c \cdot n_r + n_i - 1$ and considered as addresses which connections can be fed to. In the chromosome, each two-input node is then encoded using three integers (an address for the first input; an address for the second input; a node function). Finally, for each primary output, the chromosome contains one integer specifying the connection address. Fig. 4 shows an example and a corresponding chromosome.

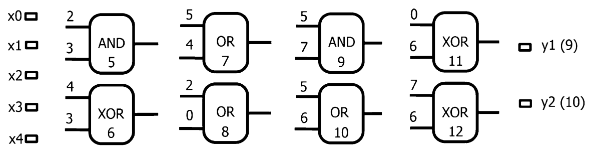

Fig. 4. Example of a combinational circuit in CGP with parameters: $n_i = 5$, $n_o = 2$, $L = 4$, $n_c = 4$, $n_r = 2$, $\Gamma = \{\text{AND (0), OR (1), XOR (2)}\}$. Gates 8, 11 and 12 are not utilized. Chromosome: 2,3,0; 4,3,2; 5,4,1; 2,0,1; 5,7,0; 5,6,1; 0,6,2; 7,6,2; 9, 10. The last two integers indicate the outputs of the circuit.

The chromosome size is $(n_a + 1)n_r n_c + n_o$ genes (integers). The main feature of this encoding is that the size of the chromosome is constant for a given n_i, n_o, n_a, n_r and n_c. However, the size of circuits represented by such chromosomes is variable as some nodes can remain disconnected. The nodes which are included into the circuit after reading the chromosome are called the active nodes.

The search is performed using a simple search strategy $(1 + \lambda)$, where λ is the number of offspring circuits created by mutation from one parent [32]. The initial population is randomly generated. A new population consisting of λ individuals is generated by applying the mutation operator on the best individual of the previous population. The mutation operator randomly modifies h integers of the chromosome. The evolution is terminated after producing a given number of generations.

In the case of combinational circuits, the fitness value of a candidate circuit is defined as [31]

$$f = \begin{cases} b & \textit{when} \quad b < n_o 2^{n_i}, \\ b + (n_c n_r - z) & \textit{otherwise}, \end{cases} \tag{1}$$

where b is the number of correct output bits obtained as response for all possible assignments to the inputs, z denotes the number of gates utilized in a particular candidate circuit and $n_c n_r$ is the total number of available gates. It can be seen that the last term $n_c n_r - z$ is considered only if the circuit behavior is perfect, i.e. $b = b_{max} = n_o 2^{n_i}$. The second term can be modified to optimize other circuit parameters.

Latency is one of the key parameters of classification. After performing numerous experiments which are reported in Section 6.2 as well as in [6], it was recognized that the minimum latency is 12Δ (where Δ is delay of a two-input gate) if fully functional coders are requested. Hence latency is not explicitly optimized in our approach; however, its maximum value is implicitly determined by $n_c = 12$.

6. Results

The experimental evaluation consists of the following steps: (1) conventional implementation of the proposed classifiers; (2) CGP-based optimization of selected subcomponents (coders); (3) resynthesis of the classifiers with optimized subcomponents; (4) verification of the quality of classification.

6.1. Conventional implementation

Three circuits corresponding to classifiers CL-acc, CL-cmp and CL-lat were behaviorally described in VHDL and synthesized into the Xilinx Virtex-7 XC7VH580T FPGA using Xilinx ISE Project navigator 14.4 tool. The target FPGA contains 6-input LUTs whose latency is 0.043 ns. The circuit latency was set as the main optimization target for the synthesis tool. Parameters of the resulting circuits which are considered as reference conventional implementations in the context of this paper are given in Table 6. One can observe

Table 6
Results of synthesis for the Xilinx Virtex-7 XC7VH580T FPGA.

Classifier	LUTs	Flip flop	Latency [ns]
CL-acc	2352	0	6.410
CL-acc + CGP	1909	0	6.113
CL-cmp	1549	0	6.093
CL-cmp + CGP	1073	0	5.604
CL-lat	1625	0	5.943
CL-lat + CGP	1217	0	5.139
Yamagaki/Clark	10,431	2326	77.504 (16 × 4.844)
AMTH	10,547	2190	71.536 (16 × 4.671)

that CL-lat is less complex and faster than CL-cmp and CL-cmp is less complex and faster than CL-acc.

6.2. Optimization by CGP

There are four types of coders in each of the three classifier circuits (Fig. 3). These 8-input/8-output coders are optimized by CGP operating at the gate level. The setting of CGP parameters is forced by the specification or it can be considered as typical for CGP. The reasons for the chosen parameter values are as follows: $n_i = 8$ and $n_o = 8$ directly follows from the specification; $n_c = 12$ reflects our strategy to restrict the maximum delay to 12Δ; $n_r = 50$ is used to provide a sufficient redundancy at the level of genotype assuming that evolved circuits will contain about 30–50 active gates [39]; L is restricted to one logic level in order to generate compact circuits and enable the deep pipeline processing in future implementations; $\lambda = 4$ is a recommended value for CGP [39,32]; and $h = 5$ corresponds with the mutation probability $5/(12 \cdot 50) = 0.00833$, i.e. with the typical values $0.001 - 0.01$ used for the mutation across almost all other studies [32]. In the case of determining the function set, we compared CGP utilizing all logic functions over two inputs except logic constants (which will be denoted Γ) against a reduced function set containing logic functions $\{a, b, \neg a, \neg b, a \vee b, a \wedge b, a \oplus b\}$. In addition to the completeness of the reduced function set (i.e. $\neg$, $\vee$ and $\wedge$ are included), the xor function ($\oplus$) is supported to enable the xor decomposition which is very useful for optimizing the xor intensive logic functions. The initial population is seeded using randomly generated circuits.

In total, circuits for 24 specifications (3 classifiers $\times$ 4 coders $\times$ 2 versions of function set) were evolved. In order to obtain basic statistics, each run consisting of 5 million generations was repeated 20 times. The number of gates, obtained at the end of CGP runs devoted to a particular specification, are presented in form of boxplots in Fig. 5 (CL-acc), Fig. 6 (CL-cmp) and Fig. 7 (CL-lat). Boxplots used in these figures contain the minimum, first quartile, median, third quartile and maximum.

The experiments confirmed our assumption that the optimized coders of CL-lat are less complex than those optimized for CL-cmp and CL-acc. It can also be seen that the usage of the complete function set consistently gives more compact coders than the reduced function set despite the fact that the search space is more complex.

In order to determine the impact of the CGP optimization to subsequent circuit synthesis and optimization conducted by means of a professional FPGA design tool, VHDL implementations of all coders evolved by CGP for CL-acc were developed and synthetized for the FPGA. The number of LUTs is presented in form of boxplots in Fig. 8. The most important observation is that the most compact FPGA implementations of coders are obtained if the circuit description entering the FPGA synthesis process contains the gates from reduced function set. It is quite unintuitive with respect to boxplots shown in Fig. 5–7. This interesting result deserves further investigations which are beyond the scope of this paper.

An analysis of optimized circuits is presented for c2 which is a middle-size coder. When optimized for the accurate CL-acc, the

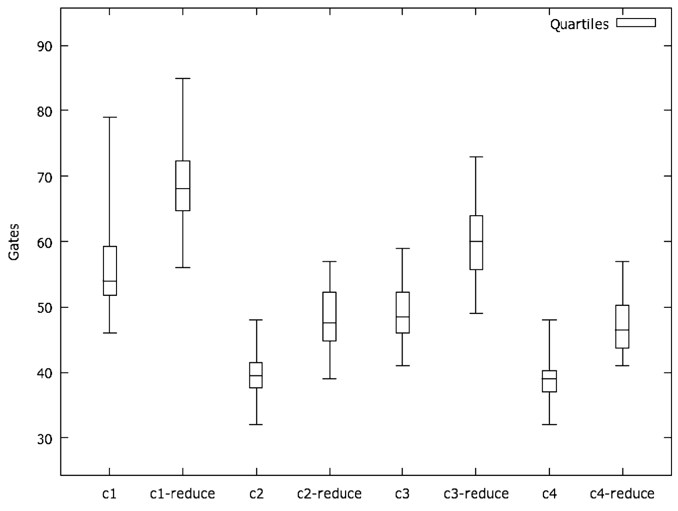

Fig. 6. The number of gates obtained at the end of 20 CGP runs for four coders of classifier CL-cmp. 'Reduce' stands for 'reduced set of gates'.

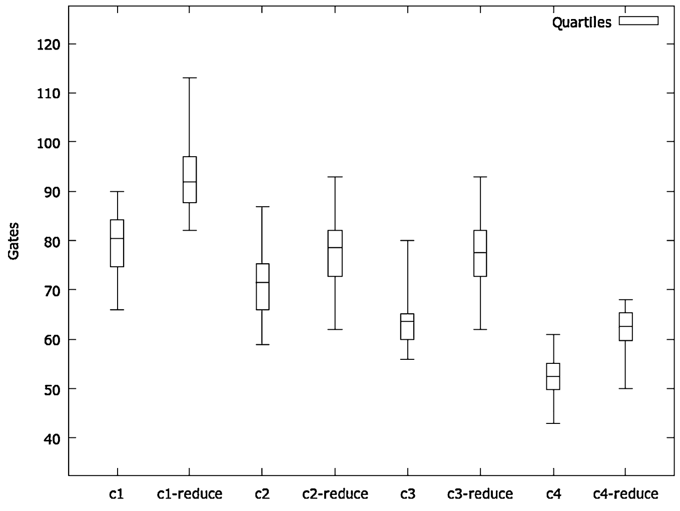

Fig. 5. The number of gates obtained at the end of 20 CGP runs for four coders (c1, c2, c3 and c4) of classifier CL-acc. 'Reduce' stands for 'reduced set of gates'.

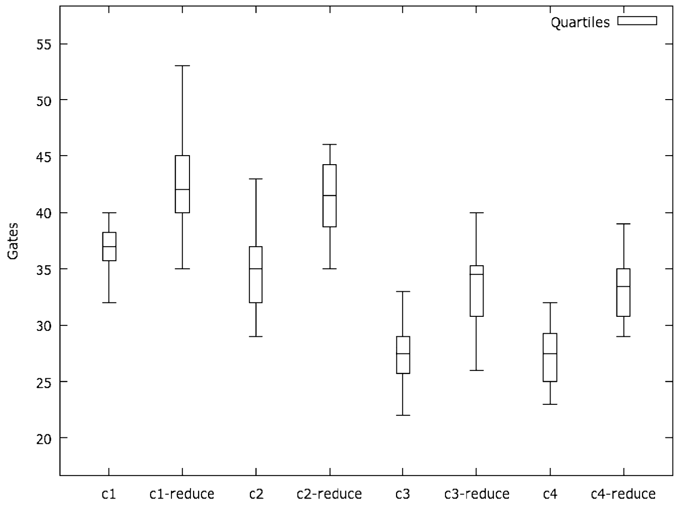

Fig. 7. The number of gates obtained at the end of 20 CGP runs for four coders of classifier CL-lat. 'Reduce' stands for 'reduced set of gates'.

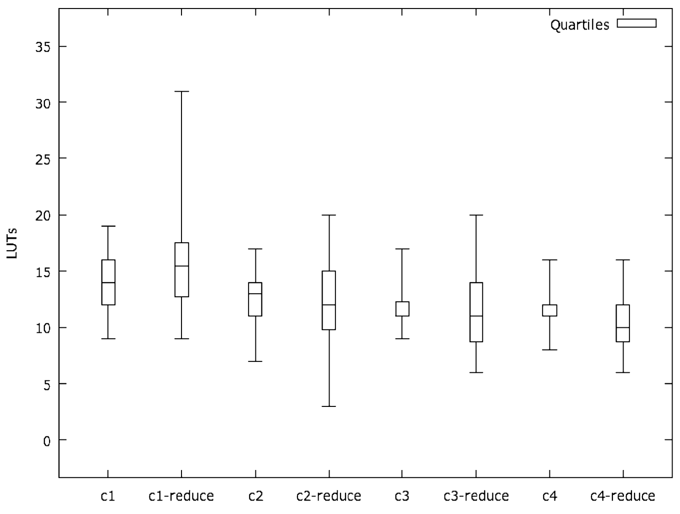

Fig. 8. The number of LUTs obtained for four coders of classifier CL-acc. 'Reduce' stands for 'reduced set of gates'.

most compact implementation of c2 consists of 59 gates (62 gates in the case of the reduced set of gates). When it is approximated for CL-lat, the most compact implementation requires only 29 gates (35 gates in the case of the reduced set of gates), which is an important reduction. At the level of LUTs, the reduction provided for CL-lat is not so remarkable because only one LUT was saved (7 versus 6 LUTs) if the FPGA synthesis starts with circuits evolved using the full gate set (Γ). However, if the FPGA synthesis starts with circuits evolved for the reduced gate set, the resulting implementation contains 3 LUTs (for CL-acc) and 2 LUTs for the most relaxed case (CL-lat). Considering the fact that 64 coders have to be implemented into the FPGA, the obtained resources reduction is significant.

6.3. Classifier resynthesis using optimized coders

The most compact implementations of coders were translated to VHDL and utilized in the VHDL code of classifiers CL-acc, CL-cmp and CL-lat. These modified classifiers were synthesized with the same setting as reported in Section 6.1.

The results of synthesis are labeled using '+CGP" and given in Table 6. Both crucial circuit parameters (latency and area expressed as the number of LUTs) were significantly improved by CGP for all classifiers.

Enabling approximate classification (CL-lat and CL-cmp) whose implementation is further optimized by CGP led to 48.2% improvement in area (LUTs) and 19.8% improvement in latency with respect to a solution (CL-acc) which would be produced by a conventional signature-based approach.

In order to compare the proposed solution with the state of the art classifiers from the literature, parameters of Yamagaki/Clark and AMTH circuit classifiers were included to Table 6. These classifiers accurately implement the L7-filter (for considered protocols) by means of optimized finite state machines. The main conclusion is that CL-lat optimized by CGP exhibits the area (LUTs) and latency one order of magnitude lower than Yamagaki/Clark and AMTH.

6.4. Quality of classification

The quality of classification was evaluated offline, utilizing a software model that has been developed for the proposed classifiers. The evaluation was performed using all three data sets in which we considered traces containing first payload packets. The output of our classifiers was verified against the L7 filter which provides 100% correct results for considered protocols. Precision and Recall metrics were calculated:

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \quad (2)$$

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \quad (3)$$

Precision informs us how many packets assigned to a given class are truly correctly assigned. Fig. 9 shows the quality of classification for the worst case – classifier CL-lat. It can be seen that HTTP, whose representation is rich in our data sets, is classified perfectly. The reason for lower percentages of Precision in the case of SMTP is the

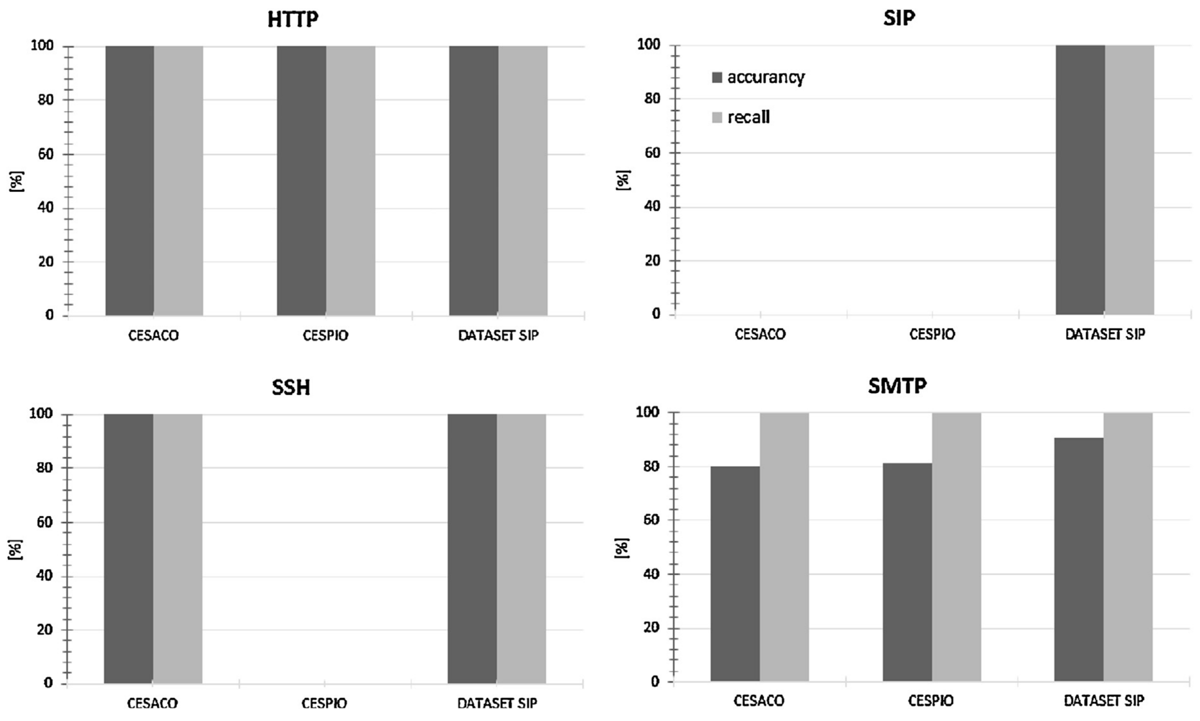

Fig. 9. Precision and Recall percentages for four classified protocols on three data sets obtained using CL-lat.

fact that considered signatures are relatively short and can easily appear inside of other protocol packets. As the subsequent packet processing is done in software precisely the incorrectly classified protocols will be recognized anyway. The software task is simpler than that of the original one. The software must only verify the labelled traffic and dismiss false positives.

Considering the whole SDM, which the proposed classifiers are targeted for, the Recall is even a more important metrics. High Recall values indicate that if a given application protocol is present in the traffic data, it is detected with almost 100% probability and thus no information is lost. Fig. 9 does not give any data for SSH in CESACO and SIP in CESACO and CESPIO. The reason is that there are no relevant records in these data sets.

7. Conclusions

It was shown how evolved circuits, optimized for the latency and area, can significantly increase the throughput of a manually designed classifier of application protocols. This paper introduced a new concept of hardware classifier which is composed as a fast combinational circuit performing signature matching where the signatures are designed according to the protocols to be classified. Its accurate implementation (CL-acc) was then relaxed and approximate classifiers CL-cmp and CL-lat were proposed with reduced area and latency. Key components of all classifiers were optimized by CGP with the aim of further area and latency reduction. This led to 48.2% improvement in area (LUTs) and 19.8% improvement in latency with respect to CL-acc. Finally, the proposed classifiers were compared with state of the art circuits accurately implementing the L7-filter and reported improvement in area and latency by one order of magnitude. The proposed solution is capable of a fast detection of key application protocols using a single packet only. It exhibits excellent Recall values (no monitored application protocols are missed). The proposed classifier will be used in the SDM framework, which will handle detailed packet processing to improve the precision parameter of the hardware classifier.

Acknowledgments

This work was supported by the Czech science Foundation project 14-04197S.

References

[1] Z. Vasicek, L. Sekanina, Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware, Genet. Program. Evol. Mach. 12 (3) (2011) 305–327.
[2] P. Kaufmann, K. Glette, T. Gruber, M. Platzner, J. Torresen, B. Sick, Classification of electromyographic signals: comparing evolvable hardware to conventional classifiers, IEEE Tran. Evol. Comput. 17 (1) (2013) 46–63.
[3] J.A. Walker, M. Trefzer, S.J. Bale, A.M. Tyrrell, Panda: a reconfigurable architecture that adapts to physical substrate variations, IEEE Trans. Comput. 62 (8) (2013) 1584–1596.
[4] L. Srivani, N.K. Giri, S. Ganesh, V. Kamakoti, Generating synthetic benchmark circuits for accelerated life testing of field programmable gate arrays using genetic algorithm and particle swarm optimization, Appl. Soft Comput. 27 (2015) 179–190.
[5] L. Kekely, J. Kucera, V. Pus, J. Korenek, A. Vasilakos, Software defined monitoring of application protocols, IEEE Trans. Comput. (2015) 1–14, http://dx.doi.org/10.1109/TC.2015.2423668.
[6] D. Grochol, L. Sekanina, M. Zadnik, J. Korenek, A fast FPGA-based classification of application protocols optimized using Cartesian GP, in: Applications of Evolutionary Computation, 18th European Conference, LNCS 9028, Springer International Publishing, 2015, pp. 67–78.
[7] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, M. Faloutsos, Is P2P dying or just hiding? in: Global Internet and Next Generation Networks, Globecom 2004, Dallas, Texas, 2004.
[8] T. Karagiannis, K. Papagiannaki, M. Faloutsos, Blinc: multilevel traffic classification in the dark SIGCOMM Comput. Commun. Rev. 35 (4) (2005) 229–240.
[9] S.-H. Yoon, J.-W. Park, J.-S. Park, Y.-S. Oh, M.-S. Kim, Internet application traffic classification using fixed IP-port, in: APNOMS, Vol. 5787 of Lecture Notes in Computer Science, Springer, 2009, pp. 21–30.
[10] A.W. Moore, D. Zuev, Internet traffic classification using Bayesian analysis techniques, in: Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS '05, ACM, 2005, pp. 50–60.
[11] S. Sen, O. Spatscheck, D. Wang, Accurate, scalable in-network identification of p2p traffic using application signatures, in: Proceedings of the 13th International Conference on World Wide Web, ACM, 2004, pp. 512–521.
[12] A. Tongaonkar, R. Keralapura, A. Nucci, Challenges in network application identification, in: Presented as Part of the 5th USENIX Workshop on Large-Scale Exploits and Emergent Threats, USENIX, Berkeley, CA, 2012.
[13] L. Bernaille, R. Teixeira, K. Salamatian, Early application identification, in: Proceedings of the 2006 ACM CoNEXT Conference, ACM, New York, NY, USA, 2006, pp. 6:1–6:12.
[14] N. Zilberman, Y. Audzevich, G. Covington, A. Moore, NetFPGA SUME: toward 100 Gbps as research commodity, Micro, IEEE 34 (5) (2014) 32–41.
[15] S. Friedl, V. Pus, J. Matousek, M. Spinler, Designing a Card for 100 Gb/s Network Monitoring, Tech. Rep., CESNET, 2013.
[16] B.-C. Park, Y. Won, M.-S. Kim, J. Hong, Towards automated application signature generation for traffic identification, in: Network Operations and Management Symposium, 2008. NOMS 2008, IEEE, 2008.
[17] V. Paxson, K. Asanović, S. Dharmapurikar, J. Lockwood, R. Pang, R. Sommer, N. Weaver, Rethinking hardware support for network analysis and intrusion prevention, in: Proceedings of the 1st USENIX Workshop on Hot Topics in Security, HOTSEC'06, USENIX Association, Berkeley, CA, USA, 2006, p. 11 http://dl.acm.org/citation.cfm?id=1268476.1268487.
[18] G. Antichi, S. Giordano, D. Miller, A. Moore, Enabling open-source high speed network monitoring on NetFPGA, in: Network Operations and Management Symposium (NOMS), 2012 IEEE, 2012, pp. 1029–1035.
[19] L. Kekely, V. Pus, P. Benacek, J. Korenek, Trade-offs and progressive adoption of FPGA acceleration in network traffic monitoring, in: 2014 24th International Conference on Field Programmable Logic and Applications (FPL), 2014, pp. 1–4.
[20] Xilinx, Ultrascale Architecture and Product Overview, 2015.
[21] L. Filtr, Project WWW Page, 2010 http://l7-filter.sourceforge.net/.
[22] R.W. Floyd, J.D. Ullman, The compilation of regular expressions into integrated circuits, J. ACM 29 (3) (1982) 603–622.
[23] R. Sidhu, V.K. Prasanna, Fast regular expression matching using FPGAs, in: FCCM '01: Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, IEEE Computer Society, 2001, pp. 227–238.
[24] C. Clark, D. Schimmel, Efficient Reconfigurable Logic circuits for matching complex network intrusion detection patterns, in: 13th International Conference on Field Programmable Logic and Application, Lisbon, Portugal, 2003, pp. 956–959.
[25] C.R. Clark, D.E. Schimmel, Scalable pattern matching for high-speed networks, in: IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), Napa, CA, 2004, pp. 249–257.
[26] S. Yun, K. Lee, Optimization of regular expression pattern matching circuit using at-most two-hot encoding on FPGA, in: International Conference on Field Programmable Logic and Applications, 2010, pp. 40–43.
[27] I. Sourdis, J. Bispo, J.M.P. Cardoso, S. Vassiliadis, Regular expression matching in reconfigurable hardware, J. Signal Process. Syst. 51 (1) (2008) 99–121.
[28] C.-H. Lin, C.-T. Huang, C.-P. Jiang, S.-C. Chang, Optimization of pattern matching circuits for regular expression on fpga, IEEE Trans. Very Large Scale Integr. Syst. 15 (12) (2007) 1303–1310.
[29] B.C. Brodie, D.E. Taylor, R.K. Cytron, A scalable architecture for high-throughput regular expression pattern matching, SIGARCH Comput. Archit. News 34 (2) (2006) 191–202.
[30] T. Higuchi, T. Niwa, T. Tanaka, H. Iba, H. de Garis, T. Furuya, Evolving hardware with genetic learning: a first step towards building a darwin machine, in: Proc. of the 2nd International Conference on Simulated Adaptive Behaviour, MIT Press, 1993, pp. 417–424.
[31] L. Sekanina, Evolvable hardware, in: Handbook of Natural Computing, Springer Verlag, 2012, pp. 1657–1705.
[32] J.F. Miller, Cartesian Genetic Programming, Springer-Verlag, 2011.
[33] E. Stomeo, T. Kalganova, C. Lambert, Generalized disjunction decomposition for evolvable hardware, IEEE Trans. Syst. Man Cybern. B 36 (5) (2006) 1024–1043.
[34] A.P. Shanthi, R. Parthasarathi, Practical and scalable evolution of digital circuits, Appl. Soft Comput. 9 (2) (2009) 618–624.
[35] P. Kaufmann, C. Plessl, M. Platzner, EvoCaches: application-specific adaptation of cache mappings, in: Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS), IEEE Computer Society, 2009, pp. 11–18.
[36] Z. Vasicek, M. Bidlo, L. Sekanina, Evolution of efficient real-time non-linear image filters for FPGAs, Soft Comput. 17 (11) (2013) 2163–2180.
[37] S.M. Cheang, K.H. Lee, K.S. Leung, Applying genetic parallel programming to synthesize combinational logic circuits, IEEE Trans. Evol. Comput. 11 (4) (2007) 503–520.
[38] H. Esmaeilzadeh, A. Sampson, L. Ceze, D. Burger, Neural acceleration for general-purpose approximate programs, Commun. ACM 58 (1) (2015) 105–115.
[39] J.F. Miller, S.L. Smith, Redundancy and computational efficiency in Cartesian genetic programming, IEEE Trans. Evol. Comput. 10 (2) (2006) 167–174.
[40] B.W. Goldman, W.F. Punch, Analysis of Cartesian genetic programming's evolutionary mechanisms, IEEE Trans. Evol. Comput. 19 (3) (2015) 359–373.