Translation Rules of Regular Expression Code for Hardware Accelerator

HENDARMAWAN[†1] Morihiro KUGA[†2] Masahiro IIDA [†2]

Abstract: In the field of industrial robotics, it is necessary to aggregate sensor information from many edge devices in an IoT environment. Therefore, in situations where real-time is required, it is necessary to perform appropriate load balancing on each tier of edge nodes, intermediate nodes, and server nodes, and to select an adequate embedded processor and to reduce network traffic between nodes. Powerful processors cannot be used on edge nodes that are required to reduce power consumption for low-cost processor. As a solution, heavy duty function of an application is effective to be implemented on FPGA as an accelerator to support the computation to address the power consumption and optimize the performance. Furthermore, real-time data streaming application become ubiquitous in the future, as industrial robotic IoT will required to process all requirement and processing into meaningful information. One of the important streaming data processing is pattern matching algorithm. There is a method to describe search pattern by a regular expression when performing data pattern matching, but it is faster to use an accelerator using hardware FPGA as IoT than using a general-purpose processor. However, it is difficult and time-consuming to design the hardware for the FPGA for each regular expression pattern. In order to improve user convenience, we are researching a method for automatically designing hardware for processing regular expressions by high-level synthesis from C language. In this research, we proposed rules and methods towards translation regular expression pattern into supported hardware code as our contribution to promote HW-SW co-design ecosystem and to allow the efficient utilization of FPGAs with low power consumption and high productivity. The performance evaluation is based on the regular expression algorithm for data streaming application on ARM processor, Core i7 CPU server and FPGA. We challenge the performance of same system of optimized C/C++ and Python Library, RE2C and our proposal rules. While, the proposed rule has been evaluated in lower cost programmable SoC device. Our result shows that it enables to speedup data streaming applications by up to 2000 and 30,000 times of C/C++ Library, 320 and 3400 times of Python Library, 15 and 180 times while compared to RE2C on CPU server and ARM respectively. In the same time reducing significantly the energy consumption with 118,000 [MB/s/J] energy efficiency.

Keywords: Industrial IoT, Stream Processing, Pattern Matching, Hardware Accelerator

1. Introduction

New trends for real-time utilization of IoT data streams become more and more common compared to data analysis on stored big data. In the future, data aggregation and processing algorithm like pattern recognition and its high speed processing for massive sensing data in Robotics IoT for Real time processing is become a must [1].

Regular expression (regex) pattern matching and feature extractions are core of big data analytics and processing real-time data from IoT for meaningful information where many implementation, rules, methodologies and surveys for IoT, Industrial IoT, data processing and analytics proposed [2]. Majority scientist and company using python as high-level programming language to perform these processes because of high productivity with rich library and support [3]. However, its performance is slow and power consumption also high. Therefore, big company recently employ FPGA accelerator to speed up and scale up performance and in the same time lower the energy consumption [4].

There is limitation when we are going to program FPGA specially to perform text pattern matching. Not to mention extra struggle for non-hardware programmer to implement regex on hardware programming language like Verilog HDL or VHDL. Thankfully, there is High Level Synthesis (HLS) to help synthesis from high level programming language C/C++ into low level programming languages to program FPGA. However, developers couldn't just implement algorithm and design pattern equipped C program into HLS directly. Some Hardware limitation apply on this condition, for example library support, dependency, and state machine wouldn't run unless some adjustment with the requirement and hardware support [5]. In this paper, we designing the rules for implementing regex patterns that can be applied to FPGA and evaluate the circuit scale and operating frequency.

A remain of this paper is organized into five sections; state of the art from past research explained in section two. The compatibility problem of High-Level Synthesis FPGA, which is have big impact on high productivity for Software and Hardware co-design thus affecting the performance of FPGA, is described in section three. The new rules of translation for C programming technique into supported HLS C is proposed. In section four, Implementation using different scenarios with CPU servers, ARM processors and FPGA accelerators, implemented on 5 different case studies will be explained. Section 5 would be evaluation of all case studies, and finally conclusion remark on section 6.

2. Related Work

Big data processing which need processing high volume of event stream in real time like finance, stock exchange, network surveillance and health-care require processing of stream event in real time scenario [6]. While software based stream monitoring has limitation due to high network packet rates as shown in [7]. Event Processing Hardware using FPGA accelerator has been

†1 Graduate School of Science and Technology, Kumamoto University, 2-39-1, Kurokami, Kumamoto, 860-8555 Japan
Email: hendarmawan@arch cs kumamoto-u ac jp,

†2 Faculty of Advanced Science and Technology, Kumamoto University, 2-39-1, Kurokami, Kumamoto, 860-8555 Japan
Email: kuga@kumamoto-u ac jp, iida@kumamoto-u ac jp

implemented by ETH Zurich University on publication [8] achieving high performance regex engine over data stream by which they have developed using VHDL low level programming. By using native design flow will enable to boost the performance, however it has drawbacks on low productivity as it required many years to developed their Pattern matching hardware accelerator due to complexity.

FPGA have demonstrated great speed performance and power efficiency advantages over conventional computers in various domains, such as image processing, communication, and data analysis. In reality, design and implementation of hardware on FPGA normally take longer period of time due to its complexity for core development and it requires understanding and skills for circuit design to use tools to develop accelerator application on FPGA. This disadvantageous attribute makes development cost on FPGA become expensive.

Pattern matching and feature extraction engine is mainly using regex. regex matching is an important mechanism used by popular network intrusion detection system (NIDS) such as Bro [9] and Snort to perform deep packet inspection against potential threats. There are few implementations on software-based regex for security like research by [10] to perform SNORT detection [11]. For hardware based, implementation for pattern matching and stream processing feature extraction have been carried out by [12] from ETH Zurich team which developing regex engines for hardware circuits using hardware description language VHDL. Another research in [13], a design, implementation and evaluation of a high-performance architecture for pattern matching is performed on FPGA to counteract the problem of increasing number of patterns to be scanned and network bottleneck. The main objective of using FPGA to solve this problem is to accelerate the system to achieve high performance.

This research is based on the evaluation from past research on data processing pattern matching using regex over software on processors and hardware FPGA. The original objectives of the authors at first, would like to knowing and investigating the different approach on high level programming library like C and Python for pattern matching software and how to improve the performance over traditional pattern matching software using hardware accelerators. Secondly, improve productivity for software and hardware developers by designing automation building block process each development flows from given pattern to code generation using high level programming language into hardware logic abstraction, design synthesis and implementation and finally generating bit-stream for hardware overlays for hardware accelerator. Third, proposing technique on the hardware design and abstraction in order to optimize and utilize hardware resources to get higher results and evaluations.

3. Proposed Rules for FPGA Pattern Matching

3.1 Problem and Challenges

There are problem and challenge to achieve our research's objectives: first, to pursue the ideal on how to get better performance over traditional pattern matching software and use lower power consumption resources in the same time. Secondly, knowing and investigating on how to improve productivity for software and hardware developers by designing automation building block process each development flows. Third, we want to optimize and utilize hardware resources by encouraging resource sharing within FPGA. Problem and challenge arises as follows:

1. Development flow of FPGA Application need complex and complicated hardware expertise, however, it guarantees better performance and low energy consumption. The initial challenge for hardware developers required to hard coded algorithm into codes and hand wire blocks of hardware programmable arrays logics from bottom – up in order to present Intellectual Property (IP) and finally utilize shells of designed vendor FPGA to be implemented on hardware level. These bottom – up principles are quite challenging and often become obstacles for software developer to design and deploy their applications on hardware level.
2. Complexity of each development process on Hardware layer and time consuming while progressing these steps are against rapid development scenario on high productivity principles. Therefore, some changes are required in order to simplify and automate the process.
3. Pattern matching software on processors are easy to use with library provided by high level programming language like C/ C++ and Python. However, performance of these implementation commonly lower compared to hard coded pattern matching software. Meanwhile, practicing pattern matching and feature extraction on processors are by far lower performances and higher energy consumption compared to hardware in chips implementation which promises higher performance and lower energy uses.

We proposed rules for designing translation for regex Hardware Accelerator aiming the high-efficiency low-cost HW/SW co-design which means short development curve and development time because we can reduce development complexity and language barrier between high-level and low-level programming languages to be synthesize on hardware level. This can be achieved by implementing code translation and linker into supported ones. After series of experiments, we developed Modification of linker RE2C [14] together with the HLS technique, it is possible to develop an accelerator in short time with an attractive acceleration performance. In addition, some optimization also provided to avoid data overhead and performing pipelines data operation.

3.2 Developing Pattern Matching and Feature Extraction

Majority of High-level programming language like C/C++, Python and Java are offering extensive library to help programmer to get easy-to-use functions and procedures to be implemented on their algorithm and application. For pattern matching algorithm for example, C/C++ provide regex library and python provided re library which can be easily deploy by importing this library into their code. There are advantages and disadvantages either using software or hardware for programmers to perform their pattern matching algorithm illustrated in figure

1. In term of productivity, computing performance, difficulties, limitation, complexity and energy consumption.

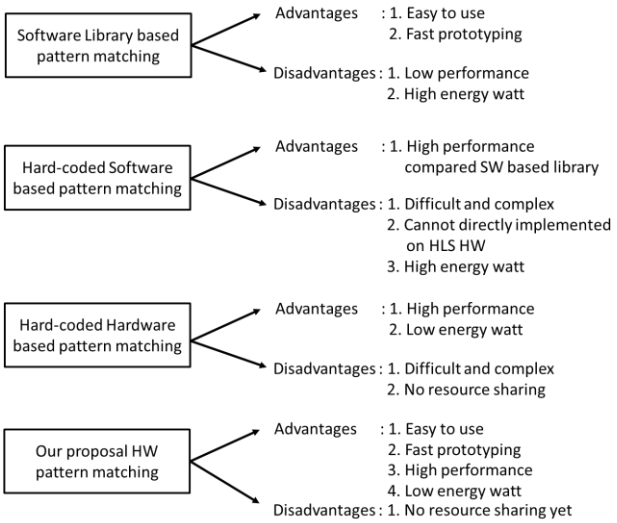

Fig.1 Different approach on software and hardware pattern matching

C/C++ and Python library enable users to use pattern of regex into coding. For C/C++ programming, it only needs to declare the header files called regex, while on Python programming, users just need to import re as described in the table.1. for the similarity between C/C++ and Python coding abstraction for pattern matching. Users then can define the need either to find first match, find all and so on. For example, frequent item counting's algorithm to collect word by word then accumulate these word pattern match and save the number of exact match over dataset.

Table.1 How to program using C++ and Python regex library

C++ Library	Python Library
`#include<regex>` `std::regex rx(R"(ABCD)");`	`import re` `regex=re.compile(r'ABCD')`

Programmer have freedom to express their way of thinking and problem solving technique which called algorithm on Software Principles. However, there will be huge challenge in order to achieve rapid development process, solving compatibility issues, trouble-shooting, resource allocation and performances for hardware implementation like FPGA which offer high performance and low energy consumption [15].

Hardware developer and vendors introduced HLS [16], state-of-art C-to-FPGA synthesis solutions in order to improved design productivity. Using HLS, help programmer to implement algorithm using C language then HLS translate it into Hardware Description Language (HDL) like VHDL and Verilog. However, C library cannot be used on High Level Synthesis (HLS) due to limitation as described on [5]. Therefore, author employ RE2C to implement pattern matching using regex on hardware level because RE2C is DFA based which suitable for programming logic FPGA. Our proposed method combines high productivity from RE2C and HLS, enable rapid development for both hardware and software developers to implement regex pattern matching accelerators on hardware.

3.3 Overview of RE2C Code generator

Regular expression to C (RE2C) is a free and open-source software laxer generator for C. Originally written by Peter Bumbulis and described in his paper, [14] it was put in public domain and has been since maintained by volunteers. It is the laxer generator adopted by projects such as PHP, Spam Assassin, ninja build system and others. Problem and challenge of using regular expression pattern for data stream applications area as follows:

1. Limited support for Hardware implementation
2. Slow performance when using C Library for Regex
3. Scarce guidelines for beginners and hardware accelerator
4. Low productivity, hard coded necessary for each step, which drawbacks for HW-SW co-design principles

In many cases, lexical analysis routines still by large using hard coded by engineer for efficiency and compatibility reason. Even though there are scanner generator available in the market like Alex [17], YAPP [18] and Mkscan [19], which can generate faster scanner than most of the hard-coded ones. However, most generated scanner is specifically targeted for a specific environment. In order to support different kind of environment it will need more effort as it is complicatedness. Fortunately, there is an open source toolkit called RE2C which proven as faster and smaller compared to those result from other scanner generator and more adaptable to other environment. The drawbacks from this scanner generator are unlike other generator, it does not provide default rules, end of input pseudo-token and buffer management routines [20] which need to be provided by users.

Internal Process of RE2C

Pattern matching using RE2C begin with scanner get input of regex then passing into parser generator, then it further processed in semantic analyzer before it optimized and generated into C++ code. Simplified of rule from RE2C are:

1. **Constructing DFA**
   First step of RE2C scanner generator is constructing DFA from pattern in order to recognize the implemented regex provided by user.
2. **Generating Code**
   After constructing DFA, the next step is parser generator, semantic analyzer generator, optimizer and code generator. Because of using DFA, code generated by RE2C relatively straightforward. It will create some additional code to save backtracking information. Example command usage:

   ```
   $ re2c –o example.c example re
   ```

   Then the code can be execute using C/C++ compiler:

   ```
   $ gcc –o executable example.c
   ```
3. **Buffering**
   RE2C generated scanner will perform checking if the buffer needed by comparing YY-CURSOR and TT-LIMIT. This attempt performed in order to reduce the amount checking. By using this effort, it will minimize steps for checking every running routine.

Authors utilize RE2C to generate minimalistic hardcoded C or C++ DFA state machine with regular expression syntax input. After that we directly compile and use the generated c code for software evaluation. However, modification and hardcoded based on RE2C generated output C code required to be able to run in HLS due to compatibility and support issues. Some rules are required in order to make HLS compatible code, in which authors proposed in the next section.

4. Translation Rules of Regular Expression for Hardware Accelerator

4.1 Into HLS C

Vivado HLS is one of the HLS tools by Xilinx in order to bridge hardware and software domain. It can help Hardware developers to work at level of abstraction while creating high performance hardware. Meanwhile, for Software developers, it can be tools to accelerate computation for their algorithm on Hardware FPGA. HLS allows them to develop algorithm in C-Level rather than hard coding in low level programming hardware languages like what Hardware programmer do in HDL language like Verilog or VHDL. Furthermore, HLS allows users to verify code and validate function more quickly than traditional DHL. It allows control C-Synthesis through optimization directive and also possible to create multiple implementation for different purposes.

4.2 Proposed Rules

As mentioned earlier, Vivado HLS does not support dynamic memory allocation, OS operation and general pointer casting. Therefore, in this paper, we proposed rules to solve this limitation over HLS, by designing lexer-adaptive of RE2C in order to create compatible C standard HLS. These rules are:

1. **Avoiding Dynamic Memory Allocation**
   Different with Static Memory Allocation, which user required to declare variables before using it so that compiler be able to allocate these variables to the memory. Dynamic Memory Allocation on the other hands, does not requires user to specify the memory allocation required for the program in advance, without worrying any upper limit of memory allocations. However, this advanced features are not supported on HLS. Therefore, the first rule is to make some adjustment for lexical analysis routine on RE2C to avoid dynamic memory allocation usage.
2. **Directing Operating System Operation into outer platform**
   With restriction all data from and to FPGA must be read and write from input and output part respectively, Operating System (OS) operation such as file read or write and OS queries like time and date are not supported by HLS. Therefore, the second rules for translation is managing OS operation into outer platform like driver and test bench in C/C++.
3. **Changing General Pointer Casting into State Machine**
   C-style pointer casts in this case are the usage of “goto” of Object-Oriented Programming approach which does not supported by HLS. Probably because of these reasons: first, it can be difficult to manage flow control. Secondly, it will lead to error prone, which can cause disaster wild pointer if excessive usage. Therefore, third rule we modified this pointer casting with simple state machine which working perfectly and more importantly it supported by HLS. Following table 2. is example of our proposed rules on this research. While RE2C is considered fastest framework for regex pattern matching compared to optimized C and Python regex library and code generator for re pattern into C. Our proposal enables to accelerate its computation on hardware level with our proposed rules.

Table.2. Rules implementation for HLS

RE2C approach	Modification based our rules

General pointer casting using ‘goto’
Example:
`yych = *++YYCURSOR;`
`switch (yych) {`
`case '@': goto yyx`
	Implement state machine like switch case
	Example:
	`case '@':`
	`inState = 2;`
	`break;`

Operating system operation:
`fr = open(argv[1], O_RDONLY);`
`rc = read(fr, buf, sizeof(buf));`
	Change into memory map and Direct memory access (DMA) on programming logic

5. Evaluation

5.1 System Setup

For implementation and evaluation, we implemented pattern matching using regex both on software and hardware approach. System setup as shown in table 3, our implementation using 3 different setups: System CPU Processor using Intel Core i7 servers DDR3 memory 16GB, ARM on PYNQ Z2 which has 650MHz dual-core Cortex-A9 ARM type processor with a DDR3 memory 512MB and ZYNQ XC7Z020-1CLG400C on PYNQ Z2 boards. Pynq is an open-source project from Xilinx that makes it easy to design embedded systems with Zynq Systems on Chips [21].

Table.3. hardware setup

Features	CPU	ARM + FPGA
Vendor	Intel CPU	PYNQ Z2
Processor	Intel Core i7 3.4GHz	A9
Cores (threads)	6 (12)	2
Architecture	64 bit	32 bit
Process	32nm	28nm
Clock Freq.	3.4 GHz	450 MHz
Level 1 cache	256 KB	32 kB
Level 2 cache	1 MB	512 kB
Level 3 cache	8 MB	-
TDP	130 W	4 W
Memory	16 GB	512 MB
OS	Ubuntu 18.04 LTS	Ubuntu18.04 LTS

5.2 Dataset

The Enron email dataset contains approximately 500,000 emails generated by employees of the Enron Corporation. It was obtained by the Federal Energy Regulatory Commission during its investigation of Enron's collapse between 1999 and 2003. This is the May 7, 2015 Version of dataset [22]. Which we divided into 10 sets of datasets for the evaluation.

5.3 Case Study and Implementation

We are using case study for most common uses of regular expression pattern matching to solve the real-world problem like explained in table 4. These five case studies also relevant with datasets, nevertheless any other regex pattern also can be applied into our proposal. From these patterns, we then perform pattern matching using different scenarios and environment in order to get a better understanding of the relation between pattern, regex library on C/C++ and Python and its performance.

Table.4. regex pattern for evaluation

No	Use Case	Regex pattern
1	Email address	`[¥w.%+-]+)@([¥w.-]+¥.[a-zA-Z]`
2	URL address	`[www]+¥.[¥w.-]+¥.[a-zA-Z]`
3	Zip code (US)	`¥s+[A-Z]{2} +¥d{5}`
4	Phone number	`[(][¥d]{3}[)][ ]?[¥d]{3}-[¥d]{4}`
5	Date	`d¥d¥s(?:Jan¦Feb¦Mar¦Apr¦May¦Jun¦Jul¦Aug¦Sep¦Oct¦Nov¦Dec)¥s¥d{4} ¥s¥d{2}:¥d{2}`

For implementation, it can be seen in table 5. There are 3 application design using C Library, Python Library and RE2C implemented on CPU server and ARM processor and proposed FPGA evaluation on PYNQ Z2 board.

Table.5. implementation methods for evaluation

Type	Methods
A1	C Library for CPU on PC
A2	C Library for ARM on PYNQ
B1	Python Library for CPU on PC
B2	Python Library for ARM on PYNQ
C1	RE2C for CPU on PC
C2	RE2C for ARM on PYNQ
D	Proposed FPGA on PYNQ

5.4 Architecture design on PYNQ

Our architecture design for hardware regex accelerators (Figure 2) designed with FPGA design Tool Vivado/Vivado HLS version 2019.1. It begins with data stream passing through PS, using C and Python API on PYNQ framework, the data then send to memory map on Processing System (PS) level. Using Python kernel and shell driver, the data on memory map connected into Direct Memory Access (DMA) on hardware FPGA, by then data to be processed on Intellectual Property (IP) on Programming Logic (PL) level. Communication vice-versa between DMA and custom IP using AXI Stream with 100MHz operating frequency and 32bit data-width. The generate result from regex IP on PL side then transported with DMA once more time to be exported to PS side using memory buffer. Finally, result pattern matching and feature extraction delivered into application like Jupyter Notebook for interactive user interface.

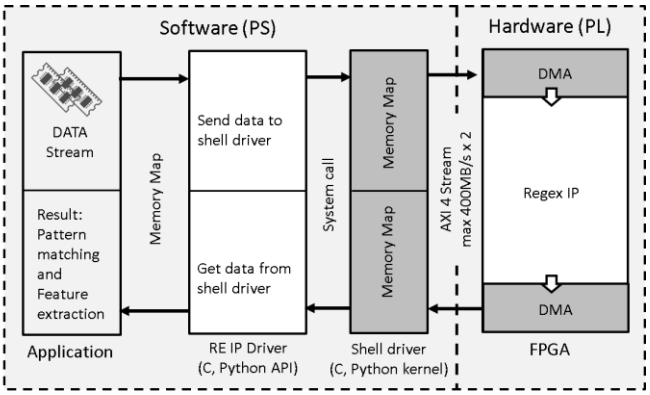

Fig.2. Architecture design on PYNQ Framework.

5.5 Result

The implementation result can be seen on table 6, where all case studies pattern matching with 10 different set of datasets implemented on CPU, ARM and FPGA. Based on evaluation, performance of regex pattern matching on ARM or CPU server using C/C++ and Python Library is much slower due to type checking and other overhead of needing to interpret code and support C/C++ and Python's abstractions. Sometimes, also the user faults also taking into consideration which called Catastrophic Backtracking which recursively backtracking for finishing line/dataset. Furthermore, the time complexity of both C++ and Python Library which using NFA lead to overhead. Instead of making O(N*M) complexity, it can be run O(2M) or worst. Meanwhile, RE2C and our approach translating NFA into DFA. It means pattern matching can be processed as linier and time complexity will be reduced significantly at O(N) complexity only, resulting higher performance.

On the other hand, Implementation using RE2C for software processors on CPU and ARM are quite fast compared to equal benchmarks with C/C++ and Python regex library as illustrated on figure 3. Finally using our rules to adapt C++ on HLS result in ultrafast regex accelerators. Our approach using optimized pre-compiled C code, is able to avoid a lot of the overhead and on HLS side, we implement optimization using data flow, pipeline and loop unroll. Another reason why our approach using hardware accelerator faster than the remaining evaluation are because of memory map and DMA to transport streaming data and then AXI 4 Stream custom IP do the computation.

The proposed hardware accelerator using proposed rules shown in table 7, dominates performance up to over 31,000 times from A2 (C/C++ Library ARM), up to 3,400 times from B2 (Python Library on ARM), up to 2,255 times compared to A1 (C/C++ Library CPU) and 321 times to B1 (Python Library CPU). Meanwhile, compared to RE2C, our implementation also dominated with up to 180 times on C2 and progressing up to 15.2 times compared to C1 despite the low hardware specifications and low power consumptions of PYNQ Z2 board.

Table. 6. Throughputs for all evaluation. [MB/s]

Method	Dataset size	1 MB	2 MB	3 MB	4 MB	5 MB	6 MB	7 MB	8 MB	9 MB	10 MB
A1 C LIBRARY on CPU	**EMAIL**	1.10	1.14	1.20	1.23	1.22	1.19	1.15	1.11	1.09	1.06
	URL	4.20	4.19	4.20	4.20	4.20	4.19	4.20	4.19	4.20	4.13
	ZIP	0.35	0.33	0.20	0.22	0.23	0.26	0.30	0.34	0.29	0.32
	PHONE	4.22	4.22	4.18	4.09	4.19	4.19	4.17	4.22	4.20	4.22
	DATE	3.80	3.77	4.05	4.01	4.08	4.07	4.06	4.06	3.94	3.61
A2 C LIBRARY on ARM	**EMAIL**	0.09	0.09	0.10	0.10	0.10	0.09	0.09	0.09	0.09	0.08
	URL	0.28	0.28	0.28	0.28	0.28	0.28	0.28	0.28	0.28	0.28
	ZIP	0.03	0.03	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02
	PHONE	0.28	0.28	0.28	0.28	0.28	0.28	0.28	0.28	0.28	0.28
	DATE	0.27	0.27	0.27	0.27	0.27	0.27	0.27	0.27	0.27	0.27
B1 PYTHON LIBRARY on CPU	**EMAIL**	11.37	8.70	8.98	10.37	9.45	9.45	10.23	9.69	9.47	9.12
	URL	41.69	37.76	31.60	38.86	43.28	46.30	38.46	41.68	40.38	41.69
	ZIP	2.55	2.36	1.39	1.50	1.52	1.78	2.10	2.37	2.04	2.27
	PHONE	142.98	153.93	157.99	160.10	156.34	146.43	140.09	140.44	134.40	135.21
	DATE	47.65	48.81	44.80	43.04	48.11	47.27	46.39	48.12	45.95	47.71
B2 PYTHON LIBRARY on ARM	**EMAIL**	0.96	0.97	1.02	1.04	1.03	1.01	1.02	1.00	0.94	0.92
	URL	2.66	3.02	3.21	3.19	2.84	2.61	2.84	2.84	2.83	2.95
	ZIP	0.22	0.21	0.13	0.14	0.15	0.17	0.20	0.22	0.19	0.21
	PHONE	10.21	10.95	11.12	11.28	11.46	11.77	11.44	11.63	11.56	11.18
	DATE	3.97	4.04	4.09	4.17	4.02	4.18	4.20	4.18	4.20	5.05
C1 RE2C on CPU	**EMAIL**	37.04	47.62	52.63	55.56	58.14	56.07	54.69	52.63	51.43	49.50
	URL	125.00	200.00	200.00	307.69	294.12	315.79	291.67	320.00	321.43	294.12
	ZIP	111.11	125.00	214.29	266.67	263.16	260.87	259.26	266.67	264.71	256.41
	PHONE	125.00	200.00	250.00	307.69	312.50	315.79	333.33	320.00	321.43	303.03
	DATE	125.00	222.22	200.00	285.71	277.78	285.71	269.23	275.86	300.00	270.27
C2 RE2C on ARM	**EMAIL**	4.44	4.74	5.04	5.10	5.10	4.91	4.76	4.64	4.46	4.19
	URL	18.18	19.23	19.61	19.80	19.92	19.93	20.00	20.10	20.13	19.96
	ZIP	17.54	18.18	18.75	17.39	18.87	17.49	18.87	18.74	18.63	18.42
	PHONE	18.18	20.62	20.98	21.28	21.37	21.35	20.71	20.73	21.58	21.65
	DATE	17.86	19.05	19.48	19.61	19.76	19.80	19.83	19.90	19.87	19.84
D PYNQ Z2	**EMAIL**	75.97	157.30	236.57	312.48	366.19	422.53	518.46	628.24	711.17	752.34
	URL	77.95	119.38	233.02	309.26	385.58	451.97	545.73	619.20	701.65	754.24
	ZIP	72.59	151.97	227.99	287.33	380.40	456.90	530.16	605.89	686.47	726.66
	PHONE	78.94	159.27	237.91	315.84	352.70	472.33	551.26	631.08	711.57	729.44
	DATE	45.32	155.81	224.01	313.94	371.11	464.14	550.43	626.13	709.30	726.03

Table. 7. Speedup PYNQ Accelerators (number of times speed-up).

	A2	**B2**	**C2**	**A1**	**B1**	**C1**
Email IP accelerator on FPGA	8,907	819	180	708	82	15.20
URL IP accelerator on FPGA	2,674	256	38	183	18	2.56
ZIP IP accelerator on FPGA	31,998	3,431	39	2,255	321	2.83
Phone IP accelerator on FPGA	2,624	65	34	173	5	2.41
Date IP accelerator on FPGA	2,700	144	37	201	15	2.69

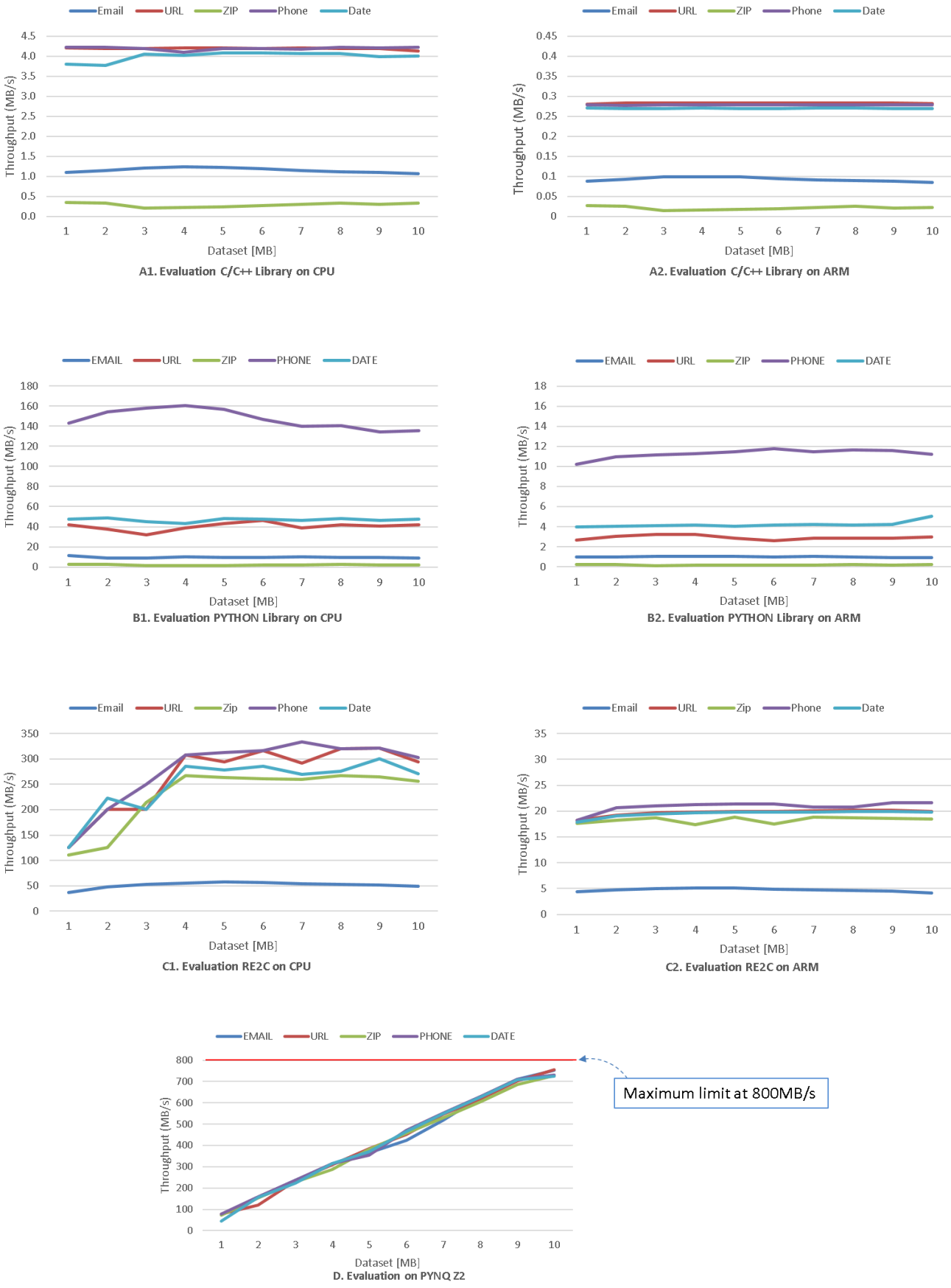

Fig.3.Throughput [MB/s] for all evaluation.

Finally, we investigated the energy consumption of CPU on PC, ARM on PYNQ and FPGA on PYNQ during the evaluations. Using equation 1 and 2, we can calculate and then evaluate the power efficiency.

$$E[\mathrm{J}] = P[W] \times t[s] \quad \ldots\ldots\ldots\ldots\ldots\ldots (1)$$

The energy E in joules [J] is equal to the average power P in watts [W], times processing time t in seconds [s]. Power is measured by USB power checker TAP-TST8 for CPU on PC and RT-USBVATM for PYNQ.

$$EE = \frac{Throughput[MB/s]}{E[\mathrm{J}]} \quad \ldots\ldots\ldots\ldots\ldots\ldots (2)$$

The energy efficiency EE is equal to the throughput [MB/s], divided by energy E in joules [J].

Table. 8. Energy efficiency evaluation

	D	A1	B1	C1
E [J]	0.01	622.78	24.44	6.84
EE [MB/s/J]	118,906	0.002	0.373	7.240
P [W]	5.03	64.9	74	43
t [s]	0.0013	9.5960	0.3303	0.1590
Throughput [MB/s]	752.34	1.06	9.12	49.5

Table 8 shows the energy consumption comparison software only and the accelerated cases between the CPU, ARM and the Zynq platform. The highest power consumption of Core i7 processors and the DRAMs is 74[W]for Python Library test with efficiency: 0.002 [MB/s/J] for C LIB, 0.373 [MB/s/J] for Python Lib, 7.240 for RE2C on CPU, meanwhile the Zynq platform (both the MPSoC FPGA and the DRAM) is 5.03 [W] with RT-USBVATM, which mean the energy consumption for accelerator is 0.01[J] and efficiency is 118,906 [MB/s/J]. This efficiency's huge advantages due to lower power consumption and fast execution time of our proposed hardware accelerator.

6. Conclusion

In this paper we presented rules and methods towards translation of regular expression pattern into supported hardware code. For evaluation we perform pattern matching and feature extraction from data stream with 5 different use-cases on 7 different implementations. We able to get better performance on Hardware Chips compared with existing high level programming common used libraries (C/C++ and Python regex library) and RE2C toolkit on embedded platform and processors. Our translation rules for hardware accelerator are proven with higher performance compared to other implementations on optimized software regex for both CPU and ARM processors by hundred and thousand times. With throughput excessing 754 [MB/s] which up to over 31,000 times better than ARM evaluation using C Library and up to 3,400 times on Python library, while compared to CPU usage both libraries up to 2,255 and 321 times speed up respectively. Next, when compared to RE2C achieved up to 180x on ARM and 15.2x on CPU. Furthermore, for energy consumption PYNQ accelerator consumed only 0.01[J] made the efficiency 118,906 [MB/s/J]. Our evaluation also showed transfer efficiency is achieved 94.3% on AXI Stream in 10MB dataset size.

For future work, we will develop hardware-software co-design framework for rapid prototyping and high productivity with evaluation on different type of FPGA. It will enable to develop real time data processing on FPGA accelerator for high performance and high energy efficiency.

Reference

[1] Yasumoto, K., Yamaguchi, H., & Shigeno, H. (2016). Survey of Real-time Processing Technologies of IoT Data Streams. J. Inf. Process., 24, 195-202.

[2] Bok, K.S., Kim, D., & Yoo, J. (2018). Complex Event Processing for Sensor Stream Data. Sensors (Basel, Switzerland), 18.

[3] McKinney, W. (2017). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython.

[4] Teubner, J., & Woods, L. (2013). Data Processing on FPGAs. Data Processing on FPGAs.

[5] Vivado Design Suite User Guide High-Level Synthesis UG902. (2019). UG902 (v2020.1) June 3, 2020.

[6] Carruthers, K. (2014). How the internet of things changes everything: The next stage of the digital revolution. Australian Journal of Telecommunications and the Digital Economy, 2, 69.

[7] Sidhu, R.P., & Prasanna, V. (2001). Fast Regular Expression Matching Using FPGAs. The 9th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'01), 227-238.

[8] Woods, L., Teubner, J., & Alonso, G. (2010). Complex event detection at wire speed with FPGAs. Proceedings of the VLDB Endowment, 3, 660 - 669.

[9] Paxson, V., Campbell, S., Leres, C., & Lee, J. (2006). Bro Intrusion Detection System.

[10] Prithi, S., Sumathi, S., & Amuthavalli, C. (2017). A Survey on Intrusion Detection System using Deep Packet Inspection for Regular Expression Matching.

[11] Shah, S.A., & Issac, B. (2018). Performance Comparison of Intrusion Detection Systems and Application of Machine Learning to Snort System. Future Gener. Comput. Syst., 80, 157-170.

[12] Woods, L., Teubner, J., & Alonso, G. (2011). Real-time pattern matching with FPGAs. 2011 IEEE 27th International Conference on Data Engineering, 1292-1295.

[13] Yang, Y., & Prasanna, V. (2012). High-Performance and Compact Architecture for Regular Expression Matching on FPGA. IEEE Transactions on Computers, 61, 1013-1025.

[14] Kiat, W.P., Mok, K.M., Lee, W., Goh, H.G., & Achar, R. (2020). An energy efficient FPGA partial reconfiguration based micro-architectural technique for IoT applications. Microprocess. Microsystems, 73, 102966.

[15] Cong, J., Liu, B., Neuendorffer, S., Noguera, J., Vissers, K., & Zhang, Z. (2011). High-Level Synthesis for FPGAs: From Prototyping to Deployment. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 30, 473-491.

[16] Bumbulis, P., & Cowan, D. (1993). RE2C: a more versatile scanner generator. LOPLAS, 2, 70-84.

[17] Mössenböck, H. (1986). Alex - A simple and efficient scanner generator. Sigplan Notices, 21, 139-148.

[18] Simões, A., Carvalho, N., & Almeida, J. (2012). Generating flex Lexical Scanners for Perl Parse: Yapp. SLATE.

[19] Horspool, R.N., & Levy, M. (1987). Mkscan - A Interactive Scanner Generator. Softw. Pract. Exp., 17, 369-378.

[20] Trofimovich, U. (2020). RE2C: A lexer generator based on lookahead-TDFA.

[21] http://www.pynq.io/

[22] https://data.world/brianray/enron-email-dataset