A Bufferless Non-exact Matching Hardware Accelerator for Processing Large Non-uniform Stream Data

Quang-Manh Duong(✉), Quang-Kien Trinh, Dinh-Ha Dao, and Trung-Nguyen

Le Quy Don Technical University, Hanoi, Vietnam
manhdq@lqdtu.edu.vn

Abstract. Recently, problems related to big data processing are becoming more and more popular and place great demands on the processing ability of the systems. The common feature of these problems is the need to find and compare data patterns in a large input data stream in real-time. Algorithms for data processing and pattern matching have been studied for a long time, including both exact and inaccurate (non-exact) solutions, at the same time, the searching data type can be uniform or heterogeneous (non-uniform). Among the proposed data processing platforms, the solution using specialized hardware accelerators proved to be superior in performance and power consumption compared to traditional solutions that combining software and the computing power of the conventional CPUs. In this study, we proposed a bufferless non-exact matching hardware accelerator for processing large non-uniform stream data on reconfigurable hardware (FPGA) combining pipeline architecture and a parallel processing approach. We analyzed the evaluation of hardware resource utilization and the data searching speed on different hardware chips, thereby giving the optimal solution for the hardware design. Finally, we practically demonstrated a design on the Kintex 7-XC7K325T FPGA device that performs pattern matching for shaping large raw input stream data. The hardware implementation from hundreds to thousands of times faster than that on software show the high applicability of the accelerator in practice.

Keywords: FPGA accelerator · Pattern matching · Parallel processing · Pipelined architecture

1 Introduction

In recent times, the problem of dealing with big data, which is prominent among these is the pattern matching problem, has become increasingly popular and has great practical value. This problem has existed for a long time, starting from the classical problems related to communication in computer networks, in which deciding the packet next hop requires reliable and timely matching (of IP/MAC and other components) algorithms. The next problems are safety and security in computer networks, requiring the system to be able to detect malicious packets, computer viruses, etc., these are the premises for

N.-S. Vo et al. (Eds.): INISCOM 2021, LNICST 379, pp. 245–258, 2021.
https://doi.org/10.1007/978-3-030-77424-0_20

the development of Network intrusion detection systems (NIDS) [1], is an important research direction in the scope of Deep Packet Investigation (DPI) [2].

Many solutions have been introduced to solve such problems. In the early stage, when there are not many options available, software solutions combined with the computing power of a shared processor (CPU) are often applied. However, the number of matching patterns was becoming larger and larger, accompanied by complex processing algorithms, making the solution using the software to face many limitations: firstly, the parallel computation on software is not effective; secondly, the bus size when using the software is fixed and small, leading to time-consuming for matching; thirdly, memory access speed on software is not high due to cumbersome architecture. Regarding the last problem, the matching process itself does not take much time, but other backend work is expensive, for example, retrieving a pattern requires a lot of memory-read operations and sometimes has to go to level catches to get data. From the aforementioned analysis, the solution using the software in combination with a conventional CPU may not be suitable for problems that require intensive or real-time computing, thus dedicated hardware solutions are a good alternative. However, hardware accelerators as well can assist but not be a substitute for the whole system, and a combination of specialized hardware and software solutions [3, 4] will be the complete approach to solve the DPI problems.

Let's take a look at a few recent case studies on pattern matching used in DPI on recently advanced hardware platforms. Wang et al. [5] proposed a pattern matching algorithm based on a skip counting automata, implemented through a three-state CAM (TCAM), which helps increasing effectiveness when the TCAM is used for regular expression-based pattern matching; Hung et al. [6] presents an efficient GPU-based multiple pattern matching algorithm for packet filtering, whereas Lin et al. [7] describe an architecture utilizing CPUs and GPUs, implementing a Length-bounded Hybrid Pattern-Matching Algorithm (LHPMA) for DPI. However, these approaches do not consider the approximate matching (they allow false positive but not the random bit error rate), while we are targeting the problem of big data where the preprocessing or data mining allows.

In the scope of this study, we primarily focus on the solution using reconfigurable hardware (FPGA) [8–12], considering their powerful computation capability and flexible customization capability, which are suitable for problems that have strict requirements in latency and bandwidth. Although the computation speed is not compared with ASIC [13], current FPGA devices are adequately powerful enough to solve problems with large bandwidth and their logical resources are sufficient for implementing massive parallel processing. The outstanding feature of FPGA compared to ASIC is that it allows for rapid development, can be easily reconfigured, and is much more cost-effective. Recent reports [14–17] show that the FPGA-based accelerators are capable to handle not only high bandwidth (i.e., up to 400 Gbps [18, 19]) for network packet parser and classification but can easily adapt to rapidly changing and development of the network protocols. The majority of prior arts are consistent in the view that classical signature-based approaches, e.g., Aho-Coasick DFA [20] in conjunction with additional techniques such as the Bloom filter (BF) [21] and Locality Hashing (LSH) [22] are extensively and practically applied [23]. Nonetheless, applying inference-based (Machine Learning) and Neural Network [24, 25] also are merely proposed, which may pave the way for revolutionalized approaches for solving modern big data problems.

In this work, we also focus on the matching problems applied for non-deterministic input stream patterns (i.e., raw binary stream) as opposed to network traffic where the flow of data is segmented/formatted into packets/frames. Besides we also consider a very practical application scenario that has not been covered in most of the previous work. First, the matching is done in a non-exact manner, i.e., the matched is considered with a certain variable permitted error threshold. Second, the design is targeted for memory-constrained devices, i.e., there is no buffering for a gigabit-level stream bandwidth but only a small on-chip cache (a few tens of Kb) serving as the interface elastic buffers[1].

The main contributions of this work are summarized as follows:

- A parallel pipelined architecture of a bufferless non-exact matching hardware accelerator for processing large non-uniform stream data on reconfigurable hardware (FPGA).
- An in-depth analysis based on implementation results on parallelism strategy and technological feasibility and limitation for applying the FPGA-based pattern matching accelerators.
- On-board demonstration of the proposed FPGA implementation using the proposed design that achieves processing speed hundreds of times higher (approximately can up to 945 times faster) on our available Digilent NETFPGA CML board (Kintex 7-XC7K325T) than on the equivalent software implementation.

The rest of this paper is organized as follows. Section 2 introduces the hardware design of the data searching accelerator. Section 3 proposes a design evaluation on dedicated reconfigurable hardware (FPGA) and gives discussions on further improvements. Section 4 presents the practical verification of the proposed design on the Kintex 7-XC7K325T FPGA device. Section 5 concludes the paper.

2 Hardware Design of Data Searching Accelerator

2.1 The Data Matching Problem

In recent years, with the development of transmission technologies, the transmission speed, as well as the data flow, have increased dramatically, so the problem of searching data in a large input data stream becomes more and more urgent. There are two main solutions in big data processing architecture: batch processing and stream-based processing, and stream-based processing is normally chosen when it is necessary to have an immediate response to the event in which data is just generated.

The requirement for big data processing problems in general, as well as the streaming processing problem is that the time of processing must be very short. The amount of data to be processed increases rapidly while the processing ability of the software is limited by the performance of the processors. The viable solution to this problem is to transfer high-speed and real-time responses to specialized hardware. Processed data will be stored for data visualization, data reporting, or data analysis later. In this work, we

[1] Due to confidential agreement, we could not go into detail or name of the specific application, but the common technical aspects are described.

designed an FPGA-based accelerator that processes and search for data from large input streaming data using pipelined architecture and parallel data processing. The problems of data searching can be solved thanks to the mentioned advantages of FPGA technology, including the working frequency is large enough to processing high-speed data flows with small latency; hardware-based parallel computing capability is the basis for data processing acceleration; flexible reconfiguration ability ensure to adapt to changes in the data flow structure.

2.2 The Architecture of the Design

In this paper, we propose a hardware design then evaluate parameters to optimize the data searching speed and resource utilization. The main part of the design is a parallel searching block called Matching Engine (ME) as shown in Fig. 1.

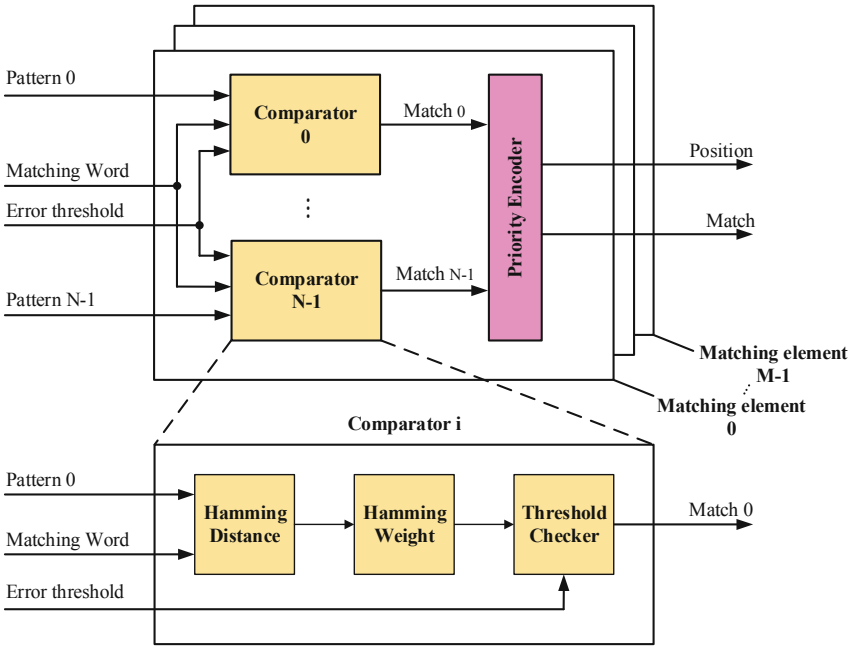


Fig. 1. Block diagram of matching engines

Input streaming data to ME is compared with pre-configured data patterns with comparators, each ME contains N such blocks. The comparator computes the required parameters such as Hamming distance, Hamming weight and compares the last parameter with the permitted deviation threshold. Finally, the Priority Encoder will decide whether to select the current data or not (by the Match signal at the output of the ME), in case the data is selected, its position in the streaming data (Position signal) will be stored. To accelerate the searching process, we arranged an array of MEs with the described working principle, i.e. having (NxM) comparators working simultaneously (in parallel).

Design of the Non-exact Comparator

For the reason that the error threshold is permitted, the comparator in this architecture is designed for signature-based matching and in the scope of circuit design. Other algorithmic enhancements are considered at other design layers. Each comparator works with two vectors of the same size x and y as shown in Fig. 2, where y is the sample binary bit string and x is the binary bit string to be compared. First, the Hamming Distance (HD) block calculates the distance between the two vectors x and the vector y, the output of this block is a binary vector, and also the input of the Hamming Weight (HW) block, which has the function of counting the number of bit 1 and the output represents the number of different positions between the original two input vectors. The comparator sets an initial fixed threshold value E (Error threshold), depending on which the comparison is classified as either an exact comparison ($E = 0$) or non-exact (inaccurate) comparison ($E \neq 0$).

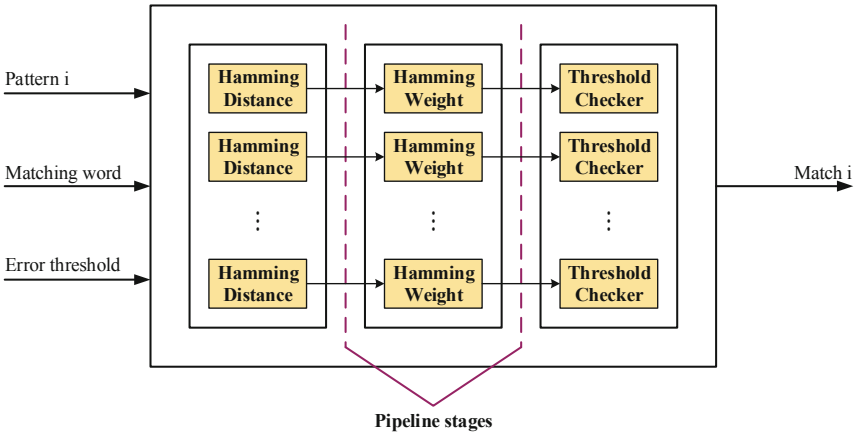


Fig. 2. Pipeline structure to calculate the hamming distance

We focused on the architectural optimizing for comparators to accelerate the computation speed for the overall ME block. In the case of exact matching, the comparator will perform a direct comparison of two input vectors, if they matched, then conclude the correct search result and vice versa, hence there is no need to use HW blocks, which greatly reduces resources and speeds up the computation. However, since the matching problem allows for a certain error threshold, in the case of non-exact matching, it is necessary to optimize the design for the bit error computation block.

Based on the above analysis, a tree adder structure capable of automatically adjusting the input data size is selected for the bit error computation. The first stage of the tree adder is optimized using the 6-input LUT of the FPGA Series 7, at the second stage, the two results are added together and moved to the following stage. This process is repeated until the final result is computed, using the inter-stage pipeline technique to ensure timing optimization with the number of the used adder stage is [log2([L/6])] in total.

The threshold checker is used to filter the input data by comparing them to a threshold value which can be set at run time. In practice, small Hamming distance values are of interest (since the input data is quite close together), large values are not considered to save processing time as well as memory during filtering the results. To ensure the best possible performance, ME and all design elements are described and optimized by the hardware description language (HDL) with the primary task of creating a customizable (extensible) design to apply for different problems. Accordingly, the parameters of the design can be generically configured. The model of the structure of comparison data and related parameters is described in Fig. 3.

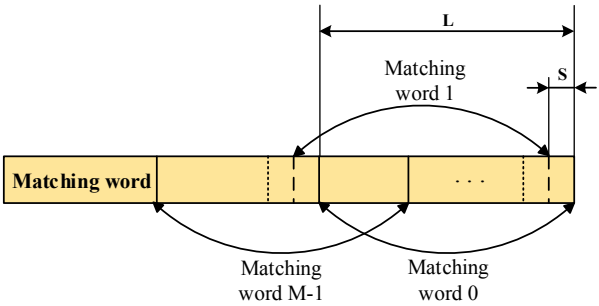


Fig. 3. Structure of the matching data

The set of design parameters includes the parameters related to the input data structure (length of matching word L, number of patterns N, length of symbol S); parameters related to the design architecture (number of matching elements M, number of pipeline stages C, system frequency F); parameter related to the searching type (exact or non-exact searching - error threshold E).

3 Design Evaluation on FPGA

We conduct the design evaluations of computation speed and resource utilization according to the parameter groups presented in Sect. 2.2, including the group of parameters involving the input data structure (L, N, S); the group of parameters related to the design architecture (M, C, F) and the parameter involving the searching method (E). These evaluations were made on the 2016.4 Xilinx Vivado software version.

3.1 Evaluation on the Impact of Parameters Related to the Input Data Structure

In this sub-section, we have chosen some representative and active FPGA families from Xilinx, including the low-cost (Artix-7), the best price/performance (Kintex-7), the performance-optimized (Virtex-7) solutions to evaluate the resource utilization when changing the length of matching word L. The graph represents the resource utilization is shown in Fig. 4 with fixed parameter set (M = 8, C = 2, F = 100 MHz).

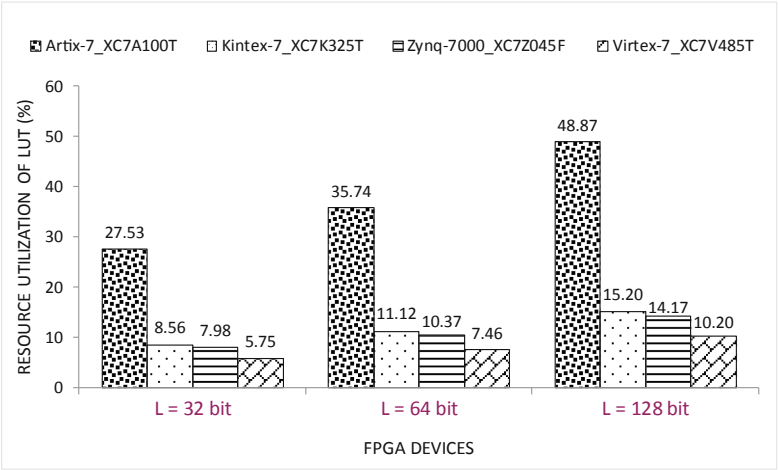


Fig. 4. Number of LUTs utilized on different FPGA devices and lengths of matching word with fixed parameter set (M = 8, C = 2, F = 100 MHz, E ≠ 0)

The evaluation results obtained show that with the same length of the matching word LUT resource utilized on different FPGA series decreases from Artix-7 to Virtex-7, which can predict through the hardware resources available on these chips announced by the manufacturer. However, the most expected result is the effect of the matching word length on resource consumption, which makes a significant difference in considered cases. For example, when the length value increased from 32 bits to 128 bits, the number of LUTs utilized in the Kintex-7 XC7K325T FPGA device almost doubled (8.56% vs 15.2%), thus reducing the length of matching word is an aspect that needs more attention.

3.2 Evaluation on the Impact of Parameters Related to the Design Architecture

Furthermore, we evaluated resource utilization on the same Kintex-7 device (XC7K325T) with different values of the number of matching elements M ranging from 2 to 32. Evaluations are conducted in both exact searching (E = 0) as well as non-exact searching (E > 0), the corresponding results of different utilized resources are shown in Fig. 5.

Theoretically, as the M value increases, the number of concurrent operations, as well as the number of logical resources utilized increases accordingly, the number of LUT utilized reaches the maximum value when the number of matching elements M = 32 (correspondingly 256 simultaneous comparisons), which accounts for less than 50%, an indication that the design is completely feasible on this FPGA platform. The analysis is evaluated in both cases of exact matching and non-exact matching.

In the case of the exact matching, there is no need to compare the results against a given error threshold, the Hamming weight block is omitted, resulting in a consequential decrease in the number of utilized logic resources. The graph showing the correlation between the percentages of utilized LUT in both cases is shown in Fig. 5. The difference in logic resource consumption is insignificant when the number of matching elements

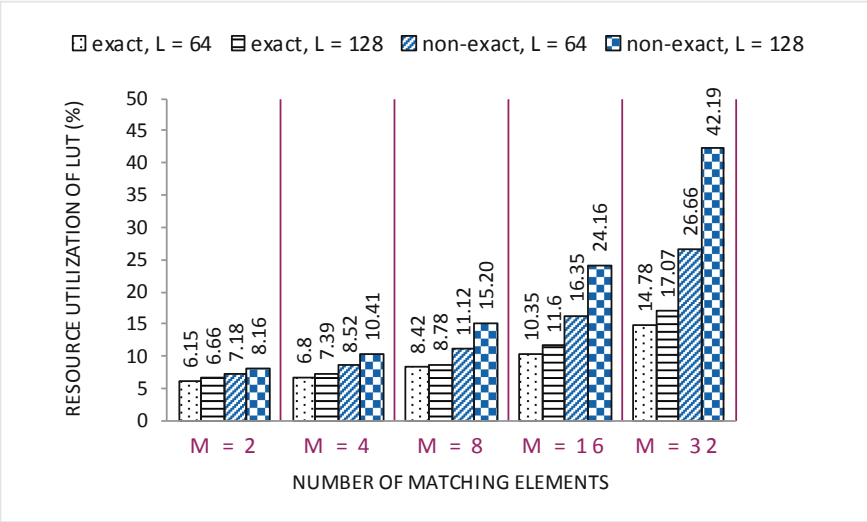


Fig. 5. The resource utilization for the cases of exact matching and non-exact matching evaluated on Kintex-7 XC7K325T with a fixed parameter set (C = 2, F = 100 MHz)

is small (M = 2; 4) and becomes quite large (about 25%) when M increases to the maximum value (M = 32). Based on the shape of the charts it is possible to comment that the LUT count grows in qua quasi-exponential manner and the parameter M has a significant influence on resource utilization.

Predictably, the resource utilized in non-exact searching is greater than in exact searching since the last one does not require the bit error counter. For example, with the same number of matching elements M = 32, non-exact searching requires 2.5 times more logical resources than an exact searching, which poses the need for optimization of the bit error calculator (HW block) to reduce the utilized hardware resources. In this study, we propose an HW block designed for the matching word of length L = 128 bits, so four design schemes for Hamming weight counter have length 128 × 1; 64 × 2; 32 × 4, and 16 × 8 respectively (see Fig. 6).

Among the designs mentioned, the one that uses four blocks with a 32-bit computational string length gives the best results, which can be explained based on the characteristics of the 7 series FPGA chips that are optimal for 6-input LUT.

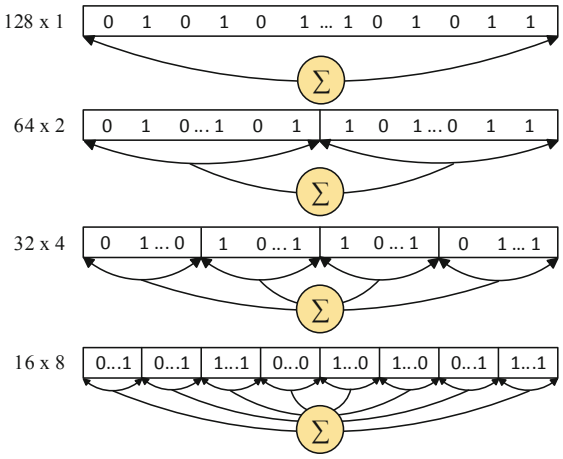


Fig. 6. Design diagram of the hamming weight counter

3.3 Performance Evaluation

The Actual Time of the Pattern Matching Process

In this section we calculate and examine the change of actual time spent on the non-exact pattern matching process, T_{actual}, calculated by the difference of theoretical time $T_{desired}$ and setup time T_{setup} (slack) on FPGA chip series. On each FPGA chip series at this time, we choose a typical device, all the time-related figures are given in Table 1.

Table 1. Actual time spent on the non-exact pattern matching process

M = 8	Artix-7	Kintex-7	Zynq-7000	Virtex-7	Kintex ultrascale
Device	XC7A100T	XC7K325T	XC7Z045F	XC7V485T	XCKU095
Actual time (ns)	13.16	9.12	8.93	9.00	6.91
Desired time (ns)	20	20	20	20	20
Slack time (ns)	6.84	10.88	11.07	11.01	13.09

Maximum Bandwidth and Comparison with Software Implementation

The data processing speed of the design (bandwidth - BW) depends on parameters including the length of symbol S, the number of matching elements M, and the system frequency F according to the formula $BW = S \times M \times F$. The maximum bandwidth of the board circuit is estimated based on the maximum value of the mentioned above parameters and depends greatly on the type of FPGA chip.

Here, we calculated the bandwidth based on the core of the design, therefore the actual value of the BW which can be achieved depends on the type of communication between

the FPGA board and the computer and this is merely a technical issue. Theoretically, the design can achieve a maximum bandwidth of up to 39.36 Gbps on the Kintex Ultrascale FPGA series (XCKU095). XC7K325T FPGA board for practical verification of the proposed design can achieve the maximum bandwidth of 10.53 Gbps corresponding to the set of parameters ($M = 48$, $L = 128$, $N = 8$, $S = 2$, $E \neq 0$), which about 945 times higher than the equivalent implementation on software, the actual bandwidth can reach a higher value by employing a larger device. The graphs of the maximum bandwidth, as well as the maximum clock frequency on the different series of FPGA, are shown in Fig. 7.

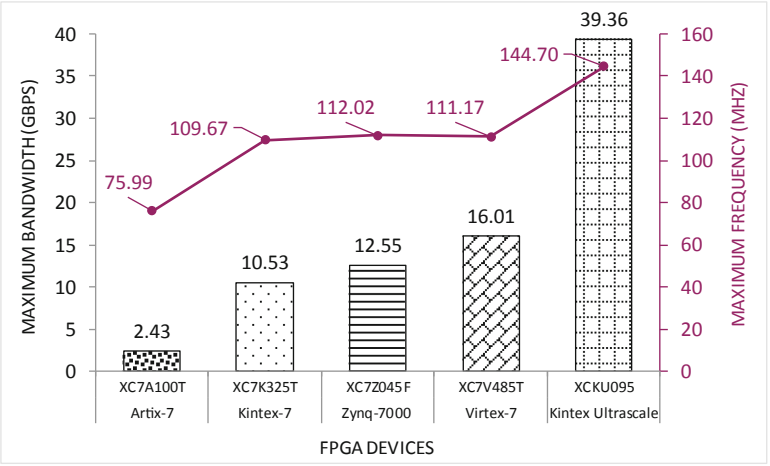


Fig. 7. The maximum bandwidth and the maximum frequency of the design on the different FPGA devices with a fixed set of parameters ($M = 48$, $L = 128$, $N = 8$, $S = 2$, $E \neq 0$)

3.4 The Multilayer Optimizing Solutions to Improve the Overall Performance of the Searching System

In this study, our solutions related to the matching core design and optimizing them for data searching belong to the under solution layer. In this solution layer, we have proposed a parallel pipelined architecture of a bufferless non-exact matching hardware accelerator, and evaluations of the impact of the various parameters on searching performance have been fully conducted in Subsects. 3.1, 3.2, and 3.3. For a general data searching problem, design proposals often target such matching cores.

However, a data searching system generally has different multiple design layers, to get the best performance, all of the design layers must be improved and optimized. Although located at different layers, the general purpose of the solutions is saving resource utilization and accelerating the data processing and in practice, these two criteria are often achieved simultaneously. Sometimes, a good solution at the upper layer, i.e. the system design level, can solve a lot of problems, including problems related to the matching core. Therefore, optimal solutions at the system design level should be

fully considered, which is also the future development direction of the research besides matching core improvement solutions.

Based on the analysis and preliminary survey of recent studies on similar problems shown in Sect. 1 of this paper, two groups of solutions are viable for the system design level, the first is the applying of classical algorithms in conjunction with additional techniques such as Bloom Filter and Locality-Sensitive Hashing to reduce the size of the data in comparison and matching processes. Some typical studies can be outlined for instance: Prya et al. [21] present a combined hierarchical approach based on an all-length Bloom filter for the source prefix field and an H-trie data structure for the destination prefix field; Lim et al. [22] extend the tuple pruning algorithm for traffic filtering; Ahmandi et al. [23] introduced a k-stage pipelined Bloom filter to improve power efficiency.

The second group of solutions deals with an area of interest recently, using Artificial Intelligence combined with Machine Learning techniques to create Neural Networks that help classify the raw (non-uniform) data from the input, which can significantly reduce the amount of data to be processed on the hardware. Neural Networks can be completely constructed on software platforms while some stages in the network requiring intensive-computation can be transferred to hardware for acceleration. Zhou et al. [24] proposed a network based on 10–30 neurons for the traffic classification task in which the difference in accuracy when using neurons 30 and 10 is negligible, compared the Naive Bayesian method with the Feed-Forward Neural Network model and found that the latter method proved more effective. Zelina et al. [25] proposed a packet-size-based classification model using early detection which classifies the flow based on the first few packets, in which the first 6 packets of the flow were used to train the NN model and classify 5 applications (SSH, Skype, HTTP, POP3, Bittorrent) with an average of 60–70% accuracy.

The solution groups related to the system design level presented above are potential options for us in improving the overall searching system, for example, hashing data before comparing, using Bloom Filters or Neural Networks to classify and reduce the size of raw data at the input of the searching system. These solutions will be considered and applied in upcoming studies.

4 The Practical Verification of the Matching Core

We finally evaluated the effectiveness of the accelerator core via a real-world application. Specifically, the core has to perform in-exact matching with a variable threshold and has matched for different types of data with a total of 512 patterns. During the practical experiment on the Kintex-7 XC7K355T FPGA device, the input of the non-uniform data flow is pushed down to the FPGA board via AXI protocol to perform the searching. The design will adjust accordingly according to the values of the general parameters outlined as mentioned above in Subsect. 2.2.

With specific design parameters, the test results show the number of utilized resources including 54% of LUT, 1% of memory elements, and 33% of RAM (Fig. 8).

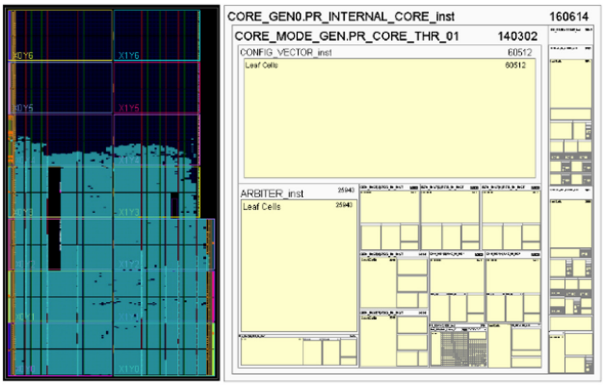
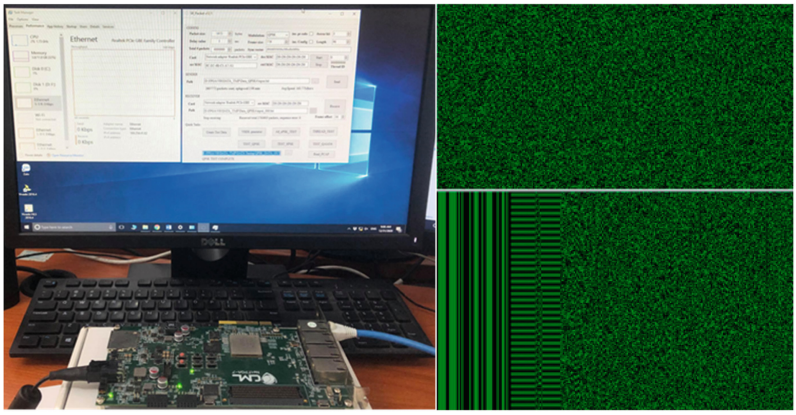


Fig. 8. Resource utilization on Kintex7-XC7K355T FPGA device for a matching core including peripherals with a fixed set of parameters (F = 100 MHz, L = 128, M = 8, N = 512, E ≠ 0)



(a) (b)

Fig. 9. Design verification on digilent NETFPGA CML board (Kintex7-XC7K355T FPGA) (a) and the bit view of the data before and after processing (b)

The process of searching the data transmitted from the computer to the FPGA board (Fig. 9a) gave accurate results, confirming the correctness of the design, but the bandwidth was limited to about 800 Mbps. This can be partly explained by the non-uniform data type at the input leading to bit-by-bit processing, and the rest due to the bandwidth limitation of the Ethernet 1 Gbps port. Figure 9b shows a bit-view image of the data before processing (top image) with a chaotic arrangement and after processing (bottom image), which has been arranged in order and formatted.

5 Conclusion

In this paper, we have proposed a design for the data matching accelerator implemented on reconfigurable hardware (FPGA) due to its outstanding technological advantages.

Design parameters were considered to evaluate the resource utilization and the data searching speed. As a case study, an implementation on the Kintex 7-XC7K325T FPGA device has been conducted, using the exact and non-exact searching method.

The solutions to improve the search speed can be using multiple parallel comparators or reducing the size of the input data, the last one is the premise for applying data hashing algorithms to reduce the size of the matching patterns, thereby speeding up the data searching on the hardware. In addition to such improvements of the hardware design, other promising solutions are the applications of classical data searching algorithms in conjunction with Bloom filter, Locality-Sensitive Hashing, or Neural Networks to reduce the size of the input data stream and we will consider their adoption for improvement of the design in the future work.

Acknowledgment. This research is funded by the Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.01–2018.310.

References

1. Huang, N.F., Hung, H.W., Lai, S.H., Chu, Y.M., Tsai, W.Y.: A GPU-based multiple-pattern matching algorithm for network intrusion detection systems. In: Proceedings of the 22nd International Conference on Advanced Information Networking and Applications (AINA), pp. 62–67 (2008)
2. Thinh, T.T., Hieu, T.N., Van Quoc, D., Kittitornkun, S.: A FPGA-based deep packet inspection engine for network intrusion detection system. In: 9th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, Phetchaburi, pp. 1–4 (2012). https://doi.org/10.1109/ECTICon.2012.6254301.
3. Fiessler, A., Hager, S., Scheuermann, B., Moore, A.W.: HyPaFilter – a versatile hybrid FPGA packet filter. In: Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems (2016)
4. Fiessler, A., Lorenz, C., Hager, S., Scheuermann, B., Moore, A.W.: HyPaFilter+: enhanced hybrid packet filtering using hardware assisted classification and header space analysis. IEEE/ACM Trans. Netw. **25**, 3655–3669 (2017)
5. Wang, F., Hong, Y., Jin, J.: Research on regular expression data packet matching algorithm based on three state content addressable memory. Int. J. Simul. – Syst. Sci. Technol. vol. **16**(5A) p. 8.1–8.5 (2015)
6. Hung, C.-L., Lin, C.-Y., Wu, P.-C.: An Efficient GPU-based multiple pattern matching algorithm for packet filtering. J. Sig. Proc. Syst. **86**(2–3), 347–358 (2016). https://doi.org/10.1007/s11265-016-1139-0
7. Lin, Y.S., Lee, C.L., Chen, Y.C.: Length-Bounded hybrid CPU/GPU pattern matching algorithm for deep packet inspection. Algorithms **10**(16), 1–13 (2017)
8. Baker, Z.K., Prasanna, V.K.: Time and area efficient pattern matching on FPGAs. In: FPGA, pp. 223–232 (2004)
9. Clark, C.R., Lee, W., Schimmel, D.E., Contis, D., Kone, M., Thomas, A.: A hardware platform for network intrusion detection and prevention. In: Proceedings of Workshop on Network Processors and Applications, pp. 136–145 (2005)
10. Clark, C.R., Schimmel, D.E.: Efficient reconfigurable logic circuits for matching complex network intrusion detection patterns. Proceedings of International Conference on Field Programmable Logic and Applications, pp. 956–959 (2003)

11. Sourdis, I., Pnevmatikatos, D.: Pre-decoded CAMs for efficient and high-speed NIDS pattern matching. In: 12th Annual IEEE FCCM, April 2004
12. Clark, C.R., Schimmel, D.E.: Scalable pattern matching for high speed networks. In: 12th Annual IEEE FCCM, April 2004
13. Liu, R.T., Huang, N.F., Chen, C.H., Kao, C.N.: A fast string-matching algorithm for network processor based intrusion detection system. ACM Trans. Embed. Comput. Syst. **3**(3), 614–633 (2004)
14. Pus, V., Kekely, L., Korenek, J.: Low-Latency modular packet header parser for FPGA. In: Proceedings of the 9th ACM/IEEE Symposium on Architecture for Networking and Communications Systems. Austin, Texas, pp. 77–78 (2012)
15. Benácek, P., Pus, V., Kubátová, H.: P4-to-VHDL: automatic generation of 100 Gbps packet parsers. In: Proceedings of the IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (2016)
16. Benácek, P., Pus, V., Korenak, J., Kekely, M.: Line rate programmable packet processing in 100Gb networks. In: Proceedings of tthe 27th International Conference on Field Programmable Logic and Applications, Ghent, Belgium (2017)
17. da Silva, J.S., Boyer, F.-R., Langlois, J.M.P.: P4-compatible high-level synthesis of low latency 100 Gb/s streaming packet parsers in FPGAs. In: Proceedings of the 26th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA 2018), February 2018. Monterey, California
18. Attig, M., Brebner, G.: 400 Gb/s programmable packet parsing on a single FPGA. In: Proceedings of the ACM/IEEE Seventh Symposium on Architectures for Networking and Communications Systems. pp. 12–23 (2011)
19. Pus, V., Kekely, L., Korenek, J.: Design methodology of configurable high performance packet parser for FPGA. In: Proceedings of the 17 International Symposium on Design and Diagnostics of Electronic Circuits and Systems. Warsaw, Poland (2014)
20. Kumar, S., Turner, J., Williams, J.: Advanced algorithm for fast and scalable deep packet inspection. In: Proceedings of the ACM/IEEE Symposium on Architecture for Networking and Communications systems. San Jose, CA (2006)
21. Priya, A.G.A., Lim, H.: Hierarchical packet classification using a bloom filter and rule-priority tries. Comput. Commun. **33**(10), 1215–1225 (2010)
22. Lim, H., Kim, S.Y.: Tuple Pruning Using Bloom Filter for Packet Classification, pp. 48–58. IEEE, Micro (2010)
23. Ahmadi, M., Wong, S.: K-Stage Pipelined bloom filter for packet classification. In: Proc. International Conference on Computational Science and Engineering. Vancouver, Canada, pp. 64–70 (2009)
24. Zhou, W., Dong, L., Bic, L., Zhou, M., Chen, L.: Internet traffic classification using feed-forward neural network. In: Proceedings of the International Conference on Computational Problem-Solving (2011)
25. Zelina, M., Oravec, M.: Early Detection of Network Applications using Neural Networks. IEEE, Elmar (2011)