YARB: a Methodology to Characterize Regular Expression Matching on Heterogeneous Systems

Filippo Carloni, Davide Conficconi, Ilaria Moschetto, Marco D. Santambrogio
Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano, Italy
{filippo.carloni, davide.conficconi, marco.santambrogio}@polimi.it; ilaria.moschetto@mail.polimi.it

***Abstract*—The continuous growth of data pushes novel and efficient approaches for information retrieval. In this context, Regular Expression (RE) matching is widely employed and represents a relevant computational kernel that carries control- and memory-related issues. Among the several solutions to relieve these burdens, accelerators seem a promising alternative to general-purpose systems. However, state-of-the-art benchmarking presents a highly fragmented scenario without consensus on the approach and lacks an open-source strategy. Therefore, to fairly characterize existing execution engines, this work presents YARB, an open benchmarking methodology. It builds upon literature solutions, a comprehensive approach, and an in-depth characterization of heterogeneous systems. Moreover, YARB's openness will enable future integrations and engines comparison.**

***Index Terms*—Regex, Benchmarking, System Evaluation**

I. Introduction and Motivation

Nowadays, given the current technological limitations [1] and the continuously growing amount of data [2], [3], *domain-specialization* represents one of major alternatives for performance improvements [4]–[8]. According to this alternative, ASIC-based Domain-Specific Architectures (DSAs) represent a mainstream strategy to tackle domain issues. However, producing a new ASIC-based DSA for each domain might not always be feasible [9]. Regular Expressions (REs), and their equivalent form of Finite State Machines (FSMs) [10], are a widely employed computational kernel [11] exploited also for efficient information retrieval. Besides being the computational core of many fields, REs are inherently sequential and may suffer from data movement bottlenecks.

Within this context, many engines, algorithms, and data structures have been proposed for efficient REs execution. CPU-based engines are among the best to handle control-intensive, variable workloads, and better fit REs matching for their software-programmability [12]–[14]. Other researchers proposed an accelerator-based model to offload REs computations from the CPUs [15], [16]. Some exploit massively parallel capabilities of GPUs [17], while others exploit the reconfigurable fabric of FPGAs for FSMs embedding [18]. Other works focus on mixing the performance of hardware-based solutions with the flexibility of CPUs, leading to hybrid DSAs implemented as ASICs or on FPGAs [19]–[22].

Despite this vast plethora of engines, there is still no consensus on a unique benchmarking methodology for the REs domain [23]–[25]. Benchmarks are essential to deliver consistent measurements of speed, efficiency, and accuracy [26]. Building on top of consistency enables dependable products, reliable comparisons, and drives innovation. Thus, many devote efforts to create consistent benchmarking. On the one hand, researchers focused on RE generators for specific domains, such as intrusion detection systems [27], which lack generality. On the other hand, utterly general-purpose benchmark generators may focus on purely theoretical aspects without strictly applying any real-case scenario [28]. The absence of a transparent approach to evaluate the REs domain hinders a clear evaluation of past, present, and future engines.

To cope with REs domain fragmentation, we propose a novel **benchmarking methodology** implemented as an **open-source evaluation suite** [29], called YARB (Yet Another Regular Expressions Benchmarking framework). We **build on top** of relevant and remarkable **past efforts** to produce a **transparent and open evaluation methodology** for the REs computational domain across heterogeneous systems. We designed YARB to be **modular and extensible** by others. YARB can exploit state-of-the-art REs and data generators [27], [28] along with proposed literature ready-to-use benchmarks [23], [24]. Since we adopt REs as universal language, we devise a novel Automata to REs translator that can optionally apply different minimization algorithms and explore solutions' performance. We analyze open-source literature for RE matching on heterogeneous systems and select the ones working on every scenario [14], [30], [31], providing relevant insights. In summary, this work's contributions are:

- A clear and novel **evaluation methodology** implemented as an **open-source** benchmark suite called YARB [29], designed to be methodologically **extensible**;
- A **systematic characterization** of REs domain with the **state-of-the-art solutions** across different **heterogeneous systems**, from embedded CPUs to data center GPUs;
- A novel extensible **Automata to REs translator** that can minimize REs, as well a characterized open dataset [29].

DOI 10.1109/ISCAS46773.2023.10181547

II. Related Work and Engines Selection

We review the REs domain literature, considering benchmarking and engines [1], detailing the one selected in §III.

A. Generators and Benchmark Suites

Since benchmarking is primarily critical to consistent work and solid foundations, many researchers focused on generators and benchmarking. For instance, Becchi et al. devised a REs generator called Regex Processor (RP) [27], but it is limited to Deep-Packet Inspection (DPI). Borsotti et al. proposed an interesting benchmark generator called REgen to evaluate advanced and ambiguous REs [28], but they evaluate mainly the tool performance for ambiguity rather than execution engines. Wadden et al. devised ANMLZoo [23], a multi-domain benchmark suite to address the literature gap in benchmarking. Unfortunately, they designed their benchmark suite to highlight the Automata Processor (AP) [32] architectural features [24] and exploiting small-scale automata [25], thus, limiting the generalization. Nourian et al. [25] tried to overcome ANMLZoo shortcomings and present a benchmark approach for large-scale automata. Unlike previous works, the authors presented end-to-end performance accounting for the whole transmission overhead. Unfortunately, they do not open their datasets and benchmarking approach. Angstadt et al. build a collection of available toolchains in the State of the Art calling this framework MNCaRT [33]. They introduce the open-source language MNRL to overcome the closed-source nature of ANML (designed for the AP) and a piece of software called `hscompile` to translate from PCRE RE format to ANML or MNRL. However, MNRL, MNCaRT, and the included tools are outdated and deprecated. Wadden et al. [24] build an improved version of ANMLZoo exploiting the MNRL language, devising open-source generators, and dropping AP-specific features. Nevertheless, they heavily rely on `hscompile`, which requires complicated setup, obsolete features, making it difficult to use nowadays.

Overall, the benchmark generators and suites have **fragmentation** in the target **scenario**, (single versus multi-domain), **language** (REs versus ANML/MNRL), and **metrics** measured.

B. Engines for Regular Expressions Matching

The vast body of research in the field of RE matching optimization confirms the community interest in several features from the efficient representation to the matching process [4], [12]–[15], [17], [20], [34], [35].

CPU-based engines: The current open-source state-of-the-art RE execution engines that exploit CPUs are Google RE2 [30] and Intel Hyperscan [14]. RE2 was designed to have predictable run-time and bounded memory consumption. Instead, Hyperscan is a high-performance engine combined with an efficient representation, which currently supports only x86 processors and focuses on DPI scenarios. These engines represent the most flexible solution to RE matching with the most substantial support, and the highest performance.

[1]It is noteworthy to remind the reader that Finite Automaton (either Deterministic/DFA or Non-Deterministic/NFA) are equivalent to a RE [10].

GPU-based engines: To improve the efficiency of RE matching, two approaches are possible. One based on multiple data streams against a single DFA (which usually exploits a depth-first execution approach); the other explores the parallel paths of an NFA (here, instead, it usually adopts a breadth-first execution model). Many works aim to exploit these parallelism degrees on top of a massively parallel architecture such as GPUs. We exclude closed-source approaches [17], [36], while selecting the approach proposed by Liu et al. [31] which exposes several state-of-the-art GPU-based methodologies.

FPGA-based engines: FPGAs are spatial architectures that enable different approaches. On the one hand, some solutions embed the automata logic directly on the reconfigurable fabric exploiting the adaptability at the cost of regenerating new bitstreams for unseen REs [16], [18], [37], [38]. On the other hand, others devise programmable DSAs to exploit the domain specialization offered by the reconfigurable hardware and expose flexibility in terms of instructions/REs to execute [22], [39], [40]. Despite some solutions are open-source, they are not documented, cumbersome to use, or do not provide means to really execute on hardware; thus, we exclude them from our work. Other open approaches present limitations in terms of searching procedure (i.e., can only find REs that match from the beginning of the data stream), such as the Vitis Library RE engine [40]. We also exclude engines limited in the amount of data able to process, such as CICERO [22]. Despite benchmarking efforts demonstrated that FPGA's spatial computing capabilities attain the highest performance at the cost of losing flexibility [25], we exclude such engines since the open-source ones do not support enough general REs features.

ASIC-based engines: This class mainly exploits the in-memory approach similar to the FPGA-embedding in different structures such as Content Addressable Memories (CAMs) [35], [41], or other special memories [19], [32], [34]. Besides, others devise DSAs, but they provide simulation results only [21]. The two primary industrial efforts from IBM with the Power Edge of Network [19] and the Micron AP [32] report extremely relevant and remarkable results. However, they are probably abandoned and unavailable; thus, we exclude them.

Despite the vast number of engines across different kinds of technologies, the off-the-shelf ones that we could exploit **experimentally** restrict to **CPU- and GPU-based engines**.

III. The YARB Methodology and Framework

To cope with REs domain fragmentation, we developed YARB, an open-source benchmarking methodology. YARB is powered by benchmark suites, generators, software, and engines available in the State of the Art coupled with custom-developed software. We design our framework to: 1) **exploit** other researchers' efforts and build on the REs as universal language; 2) **extensively** study the REs domain under different aspects and evaluate **different implications systematically** and in a **semi-automated** approach; 3) be **methodologically extensible**, **modular**, and **open** to contributions. Figure 1 represents YARB with green part coming from the literature, while the purple blocks represents our novel contributions.

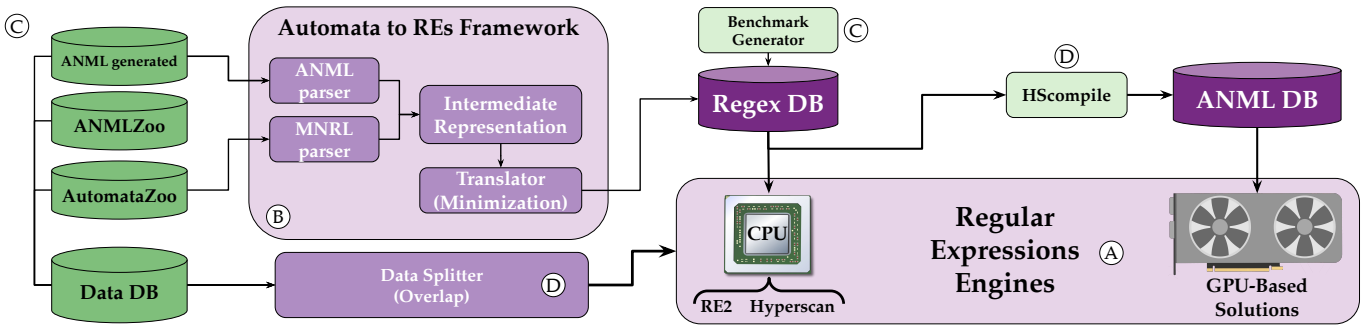

Fig. 1. YARB Framework overview. We exploit state-of-the-art benchmarks and generators (in green) to build a REs database enriched with an Automata-to-REs versions. Based on the input capabilities of the engine (e.g., RE or ANML), we analyze the execution time performance on (different) data portions.

A. Regular Expression Engines

We select the available engines as described in §II and represented in Ⓐ of Figure 1. Indeed, YARB's engines currently are: RE2 [30], Hyperscan [14], and the ones presented by Liu et al. [31]. RE2 is a C++ general-purpose library that guarantees execution time linear in the input length and fixed stack footprint, able to target any CPU regardless the ISA. Differently, Hyperscan targets only x86 CPUs and is a production-ready project. Besides, it exploits a RE decomposition algorithm to enable an efficient compilation of the REs and a SIMD-based pattern matching approach to enable multi-string matching and fast bit-based NFA. Finally, Liu et al. [31] implement several GPU-based algorithms, (that YARB all supports) including their state-of-the-art version called Obat. We select Obat since it combines a dynamic scheme resource allocation with a lookup-to-computation optimization. RE2 and Hyperscan support single- and multi-match REs, while Obat supports only multi-match. While the first two engines support RE(s)-based pattern and data stream, the GPU one works with ANML-based patterns.

B. Automata to REs framework

Since current, and likely future, engines could support natively different Automata Description Languages (ADLs), or REs, we design a novel converter from Automata to REs (A2RE) Ⓑ of Figure 1. This A2RE builds on a three-step translation process. It begins with an ADL-specific front-end that parses the input. Then, it breaks the complex multi-NFA into single automaton represented according to our Intermediate Representation (IR). Then, we can optionally minimize the IR-based Automata according to different state-of-the-art algorithms [10]. All these algorithms build on the states' reordering according to some automaton metrics and can apply both a single pass or iteratively. §IV-A characterize the minimizations, that are open for future research [29].

C. Benchmark Suite and Generators

We build a set of databases (DBs) for REs and data to analyze based on different suites and generators, represented as Ⓒ Figure 1. We build the base YARB benchmarks on top of ANMLZoo and AutomataZoo [23], [24]. These suites are composed of REs and NFA represented in ANML (MNRL) ADL(s), along with different sizes of input stimuli. We natively integrate REgen [28] as a possible source of benchmarks in YARB. We slightly modify REgen to decouple REs from the input data and use its output immediately. Moreover, we integrate benchmarks derived from the Regex Processor (RP) generator used by Liu et al. [31] as ANML inputs. We envision DBs enrichment with more benchmarks and input data stimuli.

D. New Custom Software in YARB

YARB offers an automated benchmarking flow through a set of additional software utilities, marked as Ⓓ in Figure 1.

Given the requirement of input ANML files for Obat [31] and the absence of an open standard definition, we design an adaptation flow to translate the REs into an ANML file through the only open tool, namely `hscompile` [33]. In this way, YARB supports GPU-based engines devised in [31].

Besides, RE matching is a process that is highly dependent on the target RE(s) and the input stimuli. Therefore, we design a data splitter to characterize the effect of dividing the input data into smaller chunks with a parametric overlapping factor. This component breaks down the data with a user-defined threshold and supports any size of chunking.

IV. Experimental Setup and Results

YARB and GPU handling code are in Python, while the CPU-based engines code C++. We evaluated YARB on a high-end Intel Xeon Platinum 8167M, an embedded ARM A53, and a high-end NVIDIA V100 GPU. We focus on the engines' performance for the single RE with multi-match mode.

A. Characterization of Minimization Algorithms

YARB Automata to REs framework (Figure 1 Ⓑ) implements four different minimization algorithms that reorder the automaton states according to specific features: degrees of in(out)going edges, serial-interconnection first, bridge states last, weighted according to Delgado and Morais [42]. Moreover, each reordering procedure could be repeated each time a state is eliminated (i.e., iterated or *"-i"*), or only once (i.e., *"-o"*). We characterize the performance of these algorithms considering the normalized difference (ND) in number of characters for each benchmark b with minimization m as

TABLE I
AVERAGE ND PER BENCHMARK OF YARB AUTOMATA TO REGEXES.

Minimizations	Benchmarks		
	ANMLZoo [23]	AutomataZoo [24]	Regex Processor [31]
Degree-o	0%	0.1%	0%
Degree-i	1.6%	9.6%	0%
Serial-o	19%	- †	6%
Serial-i	34%	54%	6.1%
Bridge-o	71%	91%	26.2%
Weight-o	77%	**95%**	**30%**
Weight-i	**84%**	**95%**	**30%**

† Not reported, since it attains larger REs length than baseline.

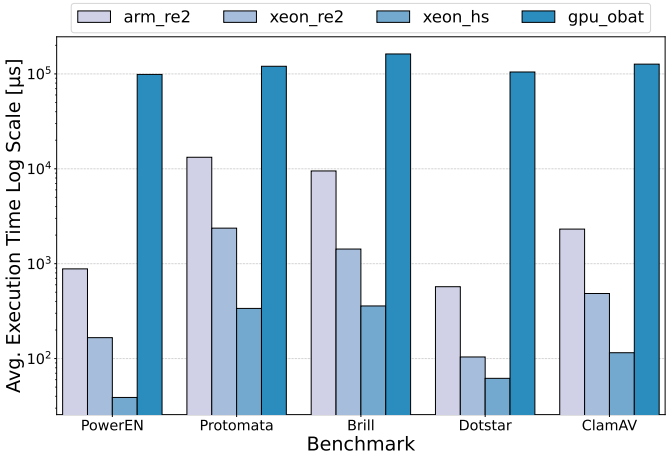

Fig. 2. Average execution time per benchmark for each execution engine in logarithmic scale (the lower, the better) of single RE multi-match mode.

$ND(b,m) = \frac{WO(b)-MIN_m(b)}{WO(b)}[\%]$, where $WO(b)$ is without, and $MIN_m(b)$ is with the benchmark minimization m applied.

Table I reports the characterization results. Our analysis showcases the weight-based minimizations as the top-performing ones across the different benchmarks and languages. *Bridge-o* reveals to be the third most performing algorithm on the evaluated datasets. Our insights suggest that the *Weight-based* minimization procedure [42] optimally combines, through the weight formula, different contributions for the state elimination procedure.

B. Comparing the Regular Expressions Engines

We now move our characterization on the analysis of the execution time automatically extracted by YARB. We analyze the single RE performance excluding any data transfer and re-iterating each test ten times to mitigate cache-warmup-related effects. Figure 2 illustrates the average execution time performance of YARB engines RE2, Hyperscan, and Obat on the corresponding architectures. The displayed average for each engine and each benchmark is computed as $Avg_{bench}(Avg_{chunk}(Exe_{time}))$, i.e., the average of the overall execution time per benchmark, considering the average execution time of the whole benchmark even when our data chunking applies. We bound our analysis to chunk sizes of

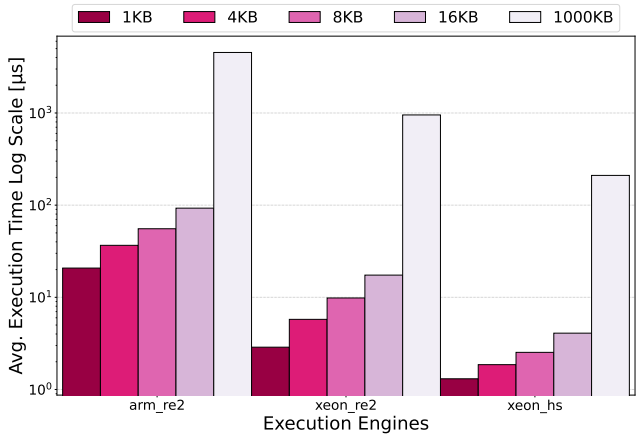

Fig. 3. Scaling of the average execution time for different single chunk sizes breakdown for ClamAV; logarithmic scale; single RE multi-match mode.

1, 4, 8, 16KB, and the whole 1MB input file to study input size variability. The figure clearly displays that Obat GPU engine does not make any sense when considering the single RE evaluation (even excluding data transfers). We compute the corresponding speedups and calculate the geomean of the speedups. Indeed, RE2 on the ARM showcases a geomean speedup of 44.5× though running on an embedded device. Finally, considering the Xeon, Hyperscan (hs in the chart) outperforms the other solutions achieving geomean speedups of 3.8×, 21.3×, 947.3× against RE2 on the Xeon, the ARM, and Obat, respectively.

C. Breakdown Chunk Analysis on Real Signature Detection

Figure 3 illustrates the scalability of Hyperscan and RE2 on the breakdown of the single execution times per single chunk of ClamAV, an ANMLZoo benchmark from a public dataset for signature detection. The engines scale sublinearly with respect to the chunk size. Therefore, whenever possible, increasing the data size processing capabilities is beneficial for CPU-based engines. Another interesting insight regards the processing time variability against the input data size. Impressively, Hyperscan demonstrates to be the engine less affected by the data sizes, possibly mitigating chunking effects.

V. FINAL REMARKS AND FUTURE WORK

This work presented YARB, an open-source benchmark suite that aims to cope with REs domain fragmentation. We designed YARB to be modular, extensible, and to exploit state-of-the-art RE engines, data generators, and benchmarks across heterogeneous systems. Since YARB adopts the REs as the universal language, we design a novel ADL-to-REs converter to unify the languages. Moreover, YARB enables a systematic characterization of matching engines, datasets, and generators in the REs domain. We believe YARB will enable fair comparisons across different possible metrics of interests.

Our next steps concern a deeper analysis of the available engines in terms of multi-REs evaluation. Hence, we will find the breakeven point where exploiting accelerators, like GPUs,

are more effective than CPUs. Finally, we strongly believe characterizing the energy efficiencies of these engines will be an essential point for YARB.

ACKNOWLEDGMENT

We would like to thank the Reviewers, E. D'Arnese, and E. Del Sozzo for the feedback. This work was supported in part by Oracle Cloud credits and related resources provided by the Oracle for Research program.

REFERENCES

[1] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture," *Communications of the ACM*, 2019.

[2] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, "Profiling a warehouse-scale computer," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, 2015, pp. 158–169.

[3] R. Gebelhoff, "Sequencing the genome creates so much data we don't know what to do with it," *The Washington Post*, pp. 1–3, 2015.

[4] K. Atasu, F. Doerfler, J. van Lunteren, and C. Hagleitner, "Hardware-accelerated regular expression matching with overlap handling on ibm poweren processor," in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IEEE, 2013, pp. 1254–1265.

[5] "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017.

[6] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. Reinhardt, A. Caulfield, E. Chung, and D. Burger, "A configurable cloud-scale dnn processor for real-time ai," in *Proceedings of the 45th International Symposium on Computer Architecture, 2018*. ACM, June 2018.

[7] E. D'Arnese, D. Conficconi, M. D. Santambrogio, and D. Sciuto, "Reconfigurable architectures: The shift from general systems to domain specific solutions," in *Emerging Computing: From Devices to Systems: Looking Beyond Moore and Von Neumann*. Springer, 2022, pp. 435–456.

[8] E. D. Sozzo, D. Conficconi, A. Zeni, M. Salaris, D. Sciuto, and M. D. Santambrogio, "Pushing the level of abstraction of digital system design: A survey on how to program fpgas," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–48, 2022.

[9] S. Hooker, "The hardware lottery," *Communications of the ACM*, 2021.

[10] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. USA: Addison-Wesley Longman Publishing Co., Inc., 2006.

[11] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis *et al.*, "The landscape of parallel computing research: A view from berkeley," *EECS Department, UC Berkeley, Tech. Rep. UCB/EECS-2006-183*, 2006.

[12] M. Becchi and P. Crowley, "An improved algorithm to accelerate regular expression evaluation," in *Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems*, 2007.

[13] J. Qiu, X. Sun, A. H. N. Sabet, and Z. Zhao, "Scalable fsm parallelization via path fusion and higher-order speculation," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 887–901.

[14] X. Wang, Y. Hong, H. Chang, K. Park, G. Langdale, J. Hu, and H. Zhu, "Hyperscan: a fast multi-pattern regex matcher for modern cpus," in *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, 2019, pp. 631–648.

[15] M. Becchi and P. Crowley, "Efficient regular expression evaluation: Theory to practice," in *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, 2008.

[16] R. Rahimi, E. Sadredini, M. Stan, and K. Skadron, "Grapefruit: An open-source, full-stack, and customizable automata processing on fpgas," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2020.

[17] E. A. Sitaridi and K. A. Ross, "Gpu-accelerated string matching for database applications," *The VLDB Journal*, 2016.

[18] T. Xie, V. Dang, J. Wadden, K. Skadron, and M. Stan, "Reapr: Reconfigurable engine for automata processing," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*.

[19] J. Van Lunteren, C. Hagleitner, T. Heil, G. Biran, U. Shvadron, and K. Atasu, "Designing a programmable wire-speed regular-expression matching accelerator," in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2012, pp. 461–472.

[20] A. Comodi, D. Conficconi, A. Scolari, and M. D. Santambrogio, "Tirex: Tiled regular expression matching architecture," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2018, pp. 131–137.

[21] H. Xia, L. Gong, C. Wang, X. Chen, and X. Zhou, "Lap: A lightweight automata processor for pattern matching tasks," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021.

[22] D. Parravicini, D. Conficconi, E. D. Sozzo, C. Pilato, and M. D. Santambrogio, "Cicero: A domain-specific architecture for efficient regular expression matching," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 5s, pp. 1–24, 2021.

[23] J. Wadden, V. Dang, N. Brunelle, T. Tracy II, D. Guo, E. Sadredini, K. Wang, C. Bo, G. Robins, M. Stan *et al.*, "Anmlzoo: a benchmark suite for exploring bottlenecks in automata processing engines and architectures," in *2016 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2016, pp. 1–12.

[24] J. Wadden, T. Tracy, E. Sadredini, L. Wu, C. Bo, J. Du, Y. Wei, J. Udall, M. Wallace, M. Stan, and K. Skadron, "AutomataZoo: A modern automata processing benchmark suite," in *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, 2018, pp. 13–24.

[25] M. Nourian, X. Wang, X. Yu, W. Feng, and M. Becchi, "Demystifying automata processing: Gpus, fpgas or micron's ap?" in *Proceedings of the International Conference on Supercomputing*. ACM, 2017.

[26] MLCommons, "Mlcommons on benchmarking," https://mlcommons.org/en/, 2022.

[27] M. Becchi, M. Franklin, and P. Crowley, "A workload for evaluating deep packet inspection architectures," in *2008 IEEE International Symposium on Workload Characterization*. IEEE, 2008, pp. 79–89.

[28] A. Borsotti, L. Breveglieri, S. C. Reghizzi, and A. Morzenti, "A benchmark production tool for regular expressions," in *International Conference on Implementation and Application of Automata*. Springer, 2019, pp. 95–107.

[29] F. Carloni, D. Conficconi, I. Moschetto, and M. D. Santambrogio, "Yarb repository," https://github.com/necst/yarb, 2023.

[30] Google, "Google re2," https://github.com/google/re2, 2020.

[31] H. Liu, S. Pai, and A. Jog, "Why gpus are slow at executing nfas and how to make them faster," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 251–265.

[32] P. Dlugosch, D. Brown, P. Glendenning, M. Leventhal, and H. Noyes, "An efficient and scalable semiconductor architecture for parallel automata processing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 12, pp. 3088–3098, 2014.

[33] K. Angstadt, J. Wadden, V. Dang, T. Xie, D. Kramp, W. Weimer, M. Stan, and K. Skadron, "MNCaRT: An open-source, multi-architecture automata-processing research and execution ecosystem," *IEEE Computer Architecture Letters*, vol. 17, no. 1, pp. 84–87, Jan 2018.

[34] V. Gogte, A. Kolli, M. J. Cafarella, L. D'Antoni, and T. F. Wenisch, "Hare: Hardware accelerator for regular expressions," in *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–12.

[35] C. R. Meiners, J. Patel, E. Norige, E. Torng, and A. X. Liu, "Fast regular expression matching using small tcams for network intrusion detection and prevention systems," in *Proceedings of the 19th USENIX conference on Security*. USENIX Association, 2010.

[36] Y. Wang, R. Watling, J. Qiu, and Z. Wang, "Gspecpal: Speculation-centric finite state machine parallelization on gpus," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2022, pp. 481–491.

[37] R. Sidhu and V. K. Prasanna, "Fast regular expression matching using fpgas," in *Field-Programmable Custom Computing Machines, 2001. FCCM'01. The 9th Annual IEEE Symposium on*. IEEE, 2001.

[38] Q. Tang, L. Jiang, X.-x. Liu, and Q. Dai, "A real-time updatable fpga-based architecture for fast regular expression matching," *Procedia Computer Science*, vol. 31, pp. 852–859, 2014.

[39] D. Conficconi, E. Del Sozzo, F. Carloni, A. Comodi, A. Scolari, and M. D. Santambrogio, "An energy-efficient domain-specific architecture for regular expressions," *IEEE Transactions on Emerging Topics in Computing*, 2022.

[40] Xilinx Inc., "Vitis Accelerated Libraries," https://github.com/Xilinx/Vitis_Libraries, 2021.
[41] J. P. C. de Lima, M. Brandalero, M. Hübner, and L. Carro, "Stap: An architecture and design tool for automata processing on memristor tcams," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 18, no. 2, pp. 1–22, 2021.
[42] M. Delgado and J. Morais, "Approximation to the smallest regular expression for a given regular language," in *International Conference on Implementation and Application of Automata*. Springer, 2004, pp. 312–314.