Enabling Efficient Regular Expression Matching at the Edge through Domain-Specific Architectures

Filippo Carloni[1], Leonardo Panseri[2], Davide Conficconi[1], Mattia Sironi[2], Marco D. Santambrogio[1]
Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano, Italy
[1]{name.lastname}@polimi.it; [2]{name.lastname}@mail.polimi.it

***Abstract*—In Cyber-Physical Systems (CPS) and Internet of Things (IoT) systems, many high-demanding applications are cut off due to the limited computational resources and the necessity to keep energy consumption as low as possible. For instance, pattern matching is a complex kernel for many essential applications, such as computer security, and the high computational and energy requirements can strongly limit its use. This paper proposes a system that combines an efficient Domain-Specific Architecture for Regular Expressions to enable host-based intrusion detection systems on edge FPGA-based devices. We demonstrate comparable execution times and remarkable energy efficiency improvements compared to a Raspberry PI. Moreover, our work aims to facilitate complex tasks offloading (such as security ones) in constrained scenarios while keeping a low energy profile and comparable performance.**

***Index Terms*—Security for IoT and CPS, DSA, Regex**

I. Introduction and Motivations

The continuous advancements of systems with reduced computational capabilities and constrained energy consumption requirements, such as Cyber-Physical Systems (CPS) and Internet of Things (IoT) systems [1], [2], have seen a surge in many fields. Indeed, industry 4.0, autonomous driving, remote surgery, and smart home systems are some of the most common examples of these systems' pervasiveness and importance [1]–[3]. These systems enhance the available functionalities by providing internet connectivity and sensing with low-priced and energy-efficient devices. Moreover, they can work in harsh environments with reduced accessibility where the devices must rely on batteries for long periods making low energy consumption paramount. The connectivity and efficiency requirements often overshadowed essential aspects, such as security. However, threats like DDoS attacks commonly exploit their pervasiveness combined with reduced security to create a widespread and distributed botnet used to attack a critical target massively [4], [5]. Furthermore, these systems' presence in critical scenarios constantly pushes security research studies that can mitigate or block attacks towards them, and botnet creation [6].

Intrusion Detection Systems (IDSs), and more specifically Host-IDSs (HIDSs), analyze network traffic and takes decisions at run-time on the device, relying on one or more malevolous patterns matched in the packets [4], [7]. However, simple string matching does not have enough expressiveness power to describe complex patterns. For this reason, security systems like pattern-matching-based HIDSs use **Regular Expressions (REs)** [4], [8]. Additionally, REs are essential in other domains like bioinformatics and Natural-Language Processing (NLP) [9]. Furthermore, REs, or the equivalent model Finite-State Automata (FSAs), are are both computational and control intensive, leading to high energy consumption and latency, that are incompatible with the requirements [10].

Meanwhile, the end of Moore's law pushes toward new architecture design approaches to reduce energy consumption, improve execution time, and reduce latency. For instance, the Domain-Specific Architectures (DSAs) promising paradigm concentrates on architectures tailored to specific domains [11]. DSAs combine the **software compilation flexibility**, essential in IDS, with the **hardware specialization**, leading to flexible and efficient solutions. An alternative to ASIC-based DSAs are the FPGA systems that enable specialized architectures with on-field programmable hardware after manufacturing, achieving remarkable efficiency and flexibility [12], [13].

For these reasons, this work builds upon CICERO [14], an open-source DSA for REs, to improve RE-based pattern matching at the edge. We detail an efficient open-source system-level methodology that leverages CICERO and novel designed components for efficient REs matching, from the hardware communication infrastructure to the software-level components. Indeed, we exploit the Arduino MKR Vidor 4000, a cheap and efficient board used in IoT and CPS systems powered by an integrated low-power Intel FPGA [15]. Overall, the main contributions are:

- A **system design methodology** of a DSA-based RE matching system with a **low-energy footprint** usable by CPS and IoT systems for **HIDS** (Section III-B1);
- The **comparison** and **results** of our **open system design**[1] against CPU-based solutions showcases top geomean improvement of **3.36**× and **15.67**× in terms of execution time and energy efficiency with state-of-the-art RE benchmarks **based on real-world use cases** (Section IV);
- An **adaptable approach** to enable computational intensive tasks **offloading** in constrained edge scenarios paving the way to future **CPS and IoT systems**.

[1]https://github.com/necst/cicero-on-vidor4000

DOI 10.1109/IPDPSW59300.2023.00023

II. CICERO Domain-Specific Architecture

We analyze the State of the Art of REs matching hardware solutions. On the one hand, many embed the REs/automata either in the reconfigurable fabric or in special memories [16], [17], while others exploit SW-programmable DSAs [14], [18]. Given the limited resources we exclude embedding solutions; therefore, we build our system design approach for HIDS on the open DSA CICERO [14] that features flexibility for pattern detection and specialization for low latency.

CICERO builds on the fundamental approach of seeing the REs as a sequence of operations executed on a sequence of characters [14], [18], [19], where it specifically execute parallel threads that working in lockstep resemble a breadth-first exploration. Moving to the kernel of CICERO, the engine is the basic unit that can perform the RE matching on an input string and returns whether it is accepted or rejected. The engine comprises a 3-stage core that performs the corresponding instruction primitive and produces the following instruction address to fetch. Then, a set of two FIFOs splits in the current character thread execution and the next thread waiting. CICERO can scale up the engine with a vectorization-like approach. This kind of engine enables a sort of window of parallel threads (or explorations) across a window of characters. In this way, this sort of parallel multi-thread execution improves the 3-stage pipelined core utilization across continuous parallel independent exploration flows. Besides, CICERO can scale out in the number of available engines with different topologies. Parravicini et al. [14] demonstrated how their FPGA-based prototype can take full advantage of a ring topology and scale the number of cores through a load-balancing infrastructure. Their evaluation on an edge device demonstrates outstanding energy efficiency making CICERO highly appealing to IoT and CPS scenarios.

III. Proposed Approach and Design Methodology

We propose a system for CPS and IoT scenarios that exploit CICERO as DSA for efficient REs execution. Section III-A describes the target device we used to deploy our final solution. Then, Section III-B displays the architectural changes and design choices we took Finally, Section III-C explains the CICERO DSA integration final execution system.

A. Arduino MKR Vidor 4000

The Arduino MKR Vidor 4000 is one of the most advanced boards in the MKR family. It combines Arduino's advantages like standardization, the high number of interfaces, and powerfulness with the Intel Cyclone LP 10CL016 FPGA chip. The board is also equipped with the ARM Cortex-M0 32-bit SAMD21 microcontroller, which, combined with the integrated low-power FPGA, can be exploited to create novel solutions in CPS and IoT systems. Specifically, the FPGA can access the I/O pins or communicate with the ARM processor to offload tasks that are otherwise too complex or not accessible with standard tiny, low-price, and energy-efficient boards.

B. Adapting CICERO to the Arduino Ecosystem

CICERO was originally designed to target AMD-Xilinx FPGAs. Though it does not embed any FPGA-part-specific macro, we tailor CICERO to target Intel FPGA technology of our system. Moreover, we design an effective communication infrastructure to connect CICERO and the microcontroller.

1) Top-Level Design Considerations: We exploited a fork of the official Arduino template [20] to ease the development process. Indeed, this template exposes all the available FPGA signals and instantiates a Phase-Locked Loop (PLL) along with an oscillator for custom clock generation. We then adapt the main interface of CICERO not to exploit any AMD-Xilinx-specific feature, such as using the AXI protocol to communicate. Hence, we design the necessary glue logic and add registers to bind CICERO top-level to our custom communication infrastructure with the microprocessor.

2) Communication Infrastructure: One of the main challenges in CICERO adaptation to these CPS and IoT scenarios has been the CPU-DSA communication. We exploit the Intel Virtual JTAG IP for simple yet effective CPU-DSA transmissions through set of Virtual Data Registers (VDRs) via the Virtual Instruction Registers (VIRs). We extend the Vidor libraries functionalities [21] with a novel library that easily handles the read and write of 32 and 64-bit registers through the Virtual JTAG protocol. Although JTAG is a serial protocol that shifts in and out bit by bit the communication data, our approach eases the CPU-DSA communication explo our library Section III-C.

3) RTL Adaptation to Quartus: Since CICERO HDL is tailored to AMD-Xilinx FPGAs, we adapted its RTL to be compatible with an Intel Cyclone 10 LP. Our adaptation can aid future work on such CPS and IoT platforms. SystemVerilog standard supported by Quartus has different limitations than the previous CICERO implementation platform. Moreover, we adapt Xilinx-specific syntax attributes to the corresponding Intel ones for example to let infer the RAM when required.

4) Bitstream Conversion: Once the overall programmable logic design is ready, we compile our design into a standard Intel FPGA bitstream. For compatibility purposes, such file must then be converted with the *vidor-bitstream-converter* utility. We exploit this C-based utility, instead of the official Arduino conversion one, to provide a streamlined flow for future work, as compiling a C program does not need any additional software on most systems.

C. System Integration Design and Implementation

We implement a reliable infrastructure to perform system validation and test real-world use cases. Figure 1 represents the whole complex system. It is composed of an external system and an Arduino running a program based on our library. Specifically, to be fully compatible with the Arduino environments and thus allow reusability for non-experts, we implement an Arduino library that configures the CICERO bitstream and provides high-level functions to control and use it. Thanks to the vast Arduino ecosystem, it is simple to integrate any external system: the Arduino Vidor board offers

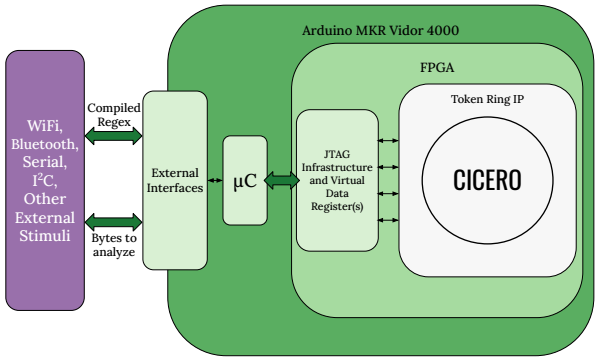


Fig. 1: Conceptual system view: the μcontroller retrieves data through our Arduino's library, and then will offload to multi-engine ring CICERO IP through the VJTAG.

a multitude of external interfaces (e.g., USB, WiFi, Bluetooth, and MiniPCI-E) that can effortlessly be used to receive data from different sources. Once retrieved the data on the Arduino, our library eases the computation offloading to the FPGA. We implement the two main components to stress-test the system and demonstrate its usefulness in real-world scenarios.

1) External System: We employ an example external Python-based host that controls the Arduino through the serial interface. It compiles an RE, transmits it, along with the data to the Arduino while waiting for results.

2) Arduino Program: The Arduino program on the device exploit our library to configure CICERO. Then, it begins to accept only REs or data to analyze as input until receiving all the bytes. The SAMD21 can also begin the execution and waits for CICERO to finish and send back the results.

Finally, we exploit an ack-based protocol to exchange information between the host and the Arduino, ensuring statu synchronization between the two main components.

IV. Experimental Setup and Results

We evaluated the proposed solution by comparing the average execution time, and the energy efficiency based on a subset of state-of-the-art high-end datacenter RE benchmarks [9], [22]. Specifically, we selected four benchmarks, spacing from security (Snort IDS from Cisco [8], a synthetic benchmark called PowerEN [9]) to other application domains, such as Protomata (bioinformatics) and Brill (NLP), to consolidate our proposed approach across various RE domains.

We use them in two modalities: *the basic mode*, which uses the RE benchmarks as-is, and *the ORed mode*, which combines multiple REs with the OR operator, similarly to Reference [14]. For each benchmark, we randomly selected 200 REs and then executed each of them on the 1MB dataset on the target systems. We measure the average execution times of 10 runs on the CPU to mitigate cold start memory effects and hardware performance counters for CICERO clock cycles.

We selected a Raspberry PI 3.B for the comparison against our solution because it represents an excellent general-purpose board that can be used in many CPS and IoT scenarios, such as HIDS [5], with a good compromise between performance and energy consumption. The board has a 1.2 GHz ARM Cortex A53 and 1 GB RAM and is configured with the Raspberry Pi OS Lite 11 (the official OS with the lowest resources and energy footprints). We select Google RE2 library [23], which is designed to have bounded memory access and predictable run-time and excluded platform-specific solutions (e.g., [24]) or those that support few REs operators. Instead, CICERO runs at 48Mhz in a 7-ring topology implementation. We measure the power consumption of the Raspberry through a Voltcraft 4000 energy logger and a USB voltemeter for the Arduino.

A. Benchmark Time and Energy Analysis

Figure 2a reports the average execution time for each benchmark executed on CICERO DSA on the Arduino and RE2 on the Raspberry in the single-RE and ORed-REs scenarios. Specifically, in simple mode, CICERO has comparable average execution times with Google RE2. Instead, in the complex scenario with multiple ORed-REs, CICERO has a remarkable speedup in almost all benchmarks. Moreover, we compute the ORed-REs speedup's overall geometric mean, and CICERO achieves a 3.36× of CICERO against RE2 on the Raspberry.

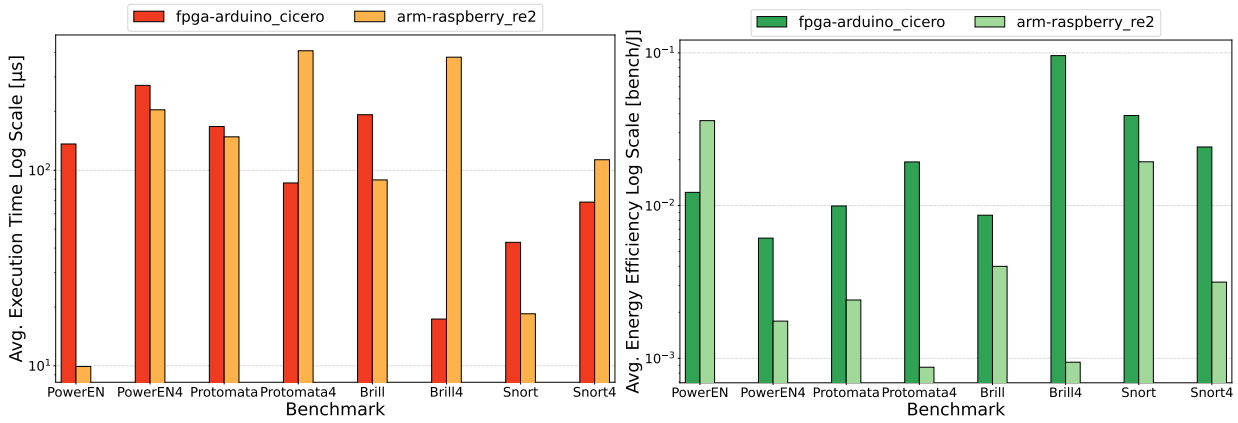
Considering the energy-efficiency results reported by Figure 2b, in the single-RE tests CICERO demonstrated good results in almost all benchmarks with a geometric mean of improvements of 1.57×. Furthermore, in the execution of multiple ORed-REs, our solution attains a geometric mean of 15.67× in energy efficiency improvement, showcasing the goodness of the proposed approach in complex benchmarks.

In conclusion, considering both the single REs and ORed REs scenarios, our system-level approach exhibits a geometric mean of 4.96× in energy efficiency improvements, confirming the goodness of the proposed approach and making it a good candidate for many CPS and IoT scenarios.

V. Related Work

Exploiting IoT and CPS systems for information and data retrieval is essential for knowledge extraction [1], and REs represent an excellent mean for pattern analysis [7]. Several approaches exist to perform RE matching efficiently (or its corresponding automata-based version) [16], [18], [24], [25]. However, the majority of the solutions target high-end, datacenter-like application scenarios, lacking an effective solution to constrained deployments like our case.

Differently, RE2 is a software library, ISA agnostic small memory footprint library [23] which can be exploited in less common and resource-limited environments. However, RE2 depends on an Operative System and might exceed small-scale IoT and CPS memory capacity. Considering HIDS, they usually provide only a software-based approach that would lock the system until the end of the computation with poor results on a Raspberry Pi [26]. Instead, offloading the HIDS pattern-matching operations to a DSA, such CICERO, could free the host and combine software programmability with a domain-specialized architecture. This combination better fits on various IoT and CPS scenarios.



(a) Execution times (the lower the better)

(b) Energy efficiencies (the higher the better)

Fig. 2: Average of matching times and energy efficiency comparison between CICERO on the Arduino MKR Vidor 4000 and the Raspberry PI ARM A53 on Simple and Ored4 benchmarks.

VI. Final Remarks

We presented a system design approach to enable complex task execution at the edge exploiting FPGA-based solutions. Our solution demonstrates an overall geometric mean improvement of 4.96× in energy efficiency against Google RE2 on a Raspberry Pi. The reported results from different real benchmarks are a concrete example of exploiting DSAs and FPGAs to bring high-demand applications in scenarios where computational power and energy consumption are limited. We hope these results can push other research works to consider our approach to removing barriers in constrained scenarios.

Acknowledgment

We would like to thank Arduino for the MKR Vidor 4000 donation, E. D'Arnese, and E. Del Sozzo for the feedback.

References

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE communications surveys & tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[2] H. Song, D. B. Rawat, S. Jeschke, and C. Brecher, *Cyber-physical systems: foundations, principles and applications*. Morgan Kaufmann, 2016.

[3] D. Cogliati, M. Falchetto, D. Pau, M. Roveri, and G. Viscardi, "Intelligent cyber-physical systems for industry 4.0," in *2018 First International Conference on Artificial Intelligence for Industries (AI4I)*. IEEE, 2018, pp. 19–22.

[4] M. F. Elrawy, A. I. Awad, and H. F. Hamed, "Intrusion detection systems for iot-based smart environments: a survey," *Journal of Cloud Computing*, vol. 7, no. 1, pp. 1–20, 2018.

[5] R. Fantacci, F. Nizzi, T. Pecorella, L. Pierucci, and M. Roveri, "False data detection for fog and internet of things networks," *Sensors*, vol. 19, no. 19, p. 4235, 2019.

[6] I. Ali, A. I. A. Ahmed, A. Almogren, M. A. Raza, S. A. Shah, A. Khan, and A. Gani, "Systematic literature review on iot-based botnet attack," *IEEE Access*, vol. 8, pp. 212 220–212 232, 2020.

[7] 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.

[8] Cisco, "Snort - open source intrusion prevention system (ips)," 2022. [Online]. Available: https://www.snort.org/

[9] 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.

[10] J. Bakker, B. Ng, W. K. Seah, and A. Pekar, "Traffic classification with machine learning in a live network," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2019, pp. 488–493.

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

[12] 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*. Springer, 2023, pp. 435–456.

[13] 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.

[14] 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*, vol. 20, no. 5s, pp. 1–24, 10 2021. [Online]. Available: https://doi.org/10.1145%2F3476982

[15] Arudino, "Mkr vidor 4000 documentation," 2022. [Online]. Available: https://docs.arduino.cc/hardware/mkr-vidor-4000

[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] 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.

[18] 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.

[19] 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.

[20] A. Williams, "Vidorfpga," https://github.com/wd5gnr/VidorFPGA, 2018.

[21] "vidor-libraries." [Online]. Available: https://github.com/vidor-libraries

[22] F. Carloni, D. Conficconi, M. Ilaria, M. D. Santambrogio *et al.*, "Yarb: a methodology to characterize regular expression matching on heterogeneous systems," in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2023, pp. 1–5.

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

[24] 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.
[25] I. Burstein, "Nvidia data center processing unit (dpu) architecture," in *2021 IEEE Hot Chips 33 Symposium (HCS)*, 2021, pp. 1–20.
[26] D. Oh, D. Kim, and W. W. Ro, "A malicious pattern detection engine for embedded security systems in the internet of things," *Sensors*, vol. 14, no. 12, pp. 24 188–24 211, 2014.

APPENDIX

A. Abstract

These Artifacts contain the code and all the necessary steps required to reproduce the execution results of CICERO on the Arduino MKR Vidor 4000, RE2 on the Raspberry Pi, and reproduce the plots of the paper.

B. Artifact check-list (meta-information)

- **Compilation:** C++, Makefile, Bash
- **Data set:** Randomly selected 200 REs from ANMLZoo and AutomataZoo benchmark suites and 1MB datasets.
- **Run-time environment:** Any OS host, Raspberry Pi Lite OS
- **Hardware:** Arduino MKR Vidor 4000, Raspberry PI 3 B;
- **Metrics:** Average Execution time in μs
- **Output:** Raw execution time results, and Charts of Figure 2
- **Experiments:** Replicate Execution times (Figure 2a), Energy Efficiency (Figure 2b) requires Figure 2a + Voltemeters
- **How much disk space required (approximately)?:** 200MB
- **How much time is needed to prepare workflow (approximately)?:** 10 minutes
- **How much time is needed to complete experiments (approximately)?:** 1 week
- **Publicly available?:** yes
- **Code licenses (if publicly available)?:** MIT License
- **Data licenses (if publicly available)?:** Refer to single benchmarks licenses.
- **Archived (provide DOI)?:** https://doi.org/10.5281/zenodo.7797406

C. Description

1) How to access: Code publicly available at https://github.com/necst/cicero-on-vidor4000/artifact_evaluation with the following DOI https://doi.org/10.5281/zenodo.7797406

2) Hardware dependencies:

- Arduino MKR Vidor 4000 board
- Requires a Raspberry PI 3 model B
- Any CPU capable of running Arduino IDE and Intel Quartus (Lite minimum)

3) Software dependencies:

- Requires Google RE2
- Requires Python 3.10 including ply, tqdm, pyserial, pandas, matplotlib, scipy, and numpy packages
- Requires screen or tmux
- Arduino IDE
- Intel Quartus Lite for bitstream regeneration

4) Data sets: The datasets are available in the ANMLZoo repository. For additional information, refer to https://github.com/necst/cicero-on-vidor4000/artifact_evaluation.

D. Installation

Clone the repository as a command or download the archive from the DOI. Install Intel Quartus Lite if you aim to regenerate the bitstream.

E. Experiment workflow

Download the archive or clone the repository

```
git clone https://github.com/necst/cicero-on-
    vidor4000.git --recurse-submodules
```

We encourage the usage of tmux or screen to have remote sessions going on, given the long time required to finish the overall execution. To program the Arduino, please follow the instruction in the *artifact-evaluation* folder.

1) RE2 Execution: Connect to the Raspberri Pi, clone the repository, and then

```
cd cicero-on-vidor4000/artifact-evaluation/
./creating_re2_benchmark.sh
```

2) CICERO Execution: Connect with a machine linked via the Micro-USB with the Arduino MKR Vidor 4000. Navigate to the folder and execute the scripts for executing all the benchmarks. Before any execution, be sure that the *arduinoport* parameter has the proper Arduino port.

```
cd cicero-on-vidor4000/cicero-arduino-drivers
./execute_all_benchmark.sh
```

This script will execute all the benchmarks and aggregate the results in a more comfortable way for reproducing the charts. Alternatively, for single benchmark execution, you may call the single benchmark script execution as `benchmark_name`:

```
./execute_{benchmark_name}.sh
```

3) Plotting: It is required to have the results all on a single machines, for instance the Raspberry Pi with the Arduino connected, and followed the previous steps. To reproduce Figure 2 then:

```
cd cicero-on-vidor4000/artifact-evaluation/plotting
./generating_folders_structure.sh
python charts.py 0
python charts.py 1
```

F. Evaluation and expected results

These Artifacts mainly replicate Figure 2a of the execution times from scratch on the Arduino MKR Vidor 4000 and the Raspberry PI. We included the power data to replicate Figure 2b. Collecting them from scratch requires physical access to the devices and the voltmeters. The artifacts let reproduce both numerical and graphical representation of the manuscript.

G. Experiment customization

To customize CICERO bitstream, you can find the Quartus Lite project under the `<top_folder>/projects/cicero-cyclone/MKRVIDOR4000qpf`, customize the project in any way, and finally generate the bitstream. Follow the instructions in the "About vidor-bitstream-converter" to convert your bitstream, and then replace the novel generated `app.h`.

To customize the experiments, the user must first pick the dataset (REs and input data), and dispose them as for the other benchmarks. Then follow the instruction to run.

H. Methodology

Submission, reviewing and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-badging
- http://cTuning.org/ae/submission-20201122.html
- http://cTuning.org/ae/reviewing-20201122.html