Investigating the performance of MicroPython and C on ESP32 and STM32 microcontrollers

Valeriu Manuel Ionescu
Dept. Electronics, Communications and Computers
University of Pitesti
Romania
manuelcore@yahoo.com

Florentina Magda Enescu
Dept. Electronics, Communications and Computers
University of Pitesti
Romania
enescu_flor@yahoo.com

***Abstract*—Python is a programming language that is used both by entry level programmers and advanced researchers. MicroPython is a software implementation of Python that runs on microcontrollers. This paper will investigate the MicroPython execution performance compared to similar C native code on low cost microcontrollers: STM32 and ESP32. The comparison will target: memory allocation speed; SHA-256 and CRC-32 performance and will present conclusions regarding the encountered problems and ways to improve the application performance.**

Keywords—MicroPython, ESP32, STM32, performance, hash, CRC

I INTRODUCTION

Edge computing is a term associated to Internet of Things (IoT) devices that identifies the partial or complete processing of data captured from sensors on the system that makes the data acquisition. This is possible now because the processing power of the microcontrollers [9] has increased considerably while maintaining an acceptable price. While C code on these devices is preferable, having a larger codebase and a large number of programmers is also desirable, therefore being able to write code fast with as many coders as possible makes investigating the performance of Python code on microcontrollers a priority.

Python is a high level programming language that is often used in website development, GUI applications, machine learning and Internet of Things (IoT) applications. Some of its advantages are the ease of code writing and reading, running the same code (with no need to recompile) on multiple platforms as it is an interpreted programming language, the automatic memory management and the large number of libraries that allows developing prototype applications fast.

There are two major versions of Python that are used for development: 2 and 3. While the user base seemed to be split between the two versions, there is a rapid transition towards version 3 [1] [2] of more than 75% of the coders. Finally an announcement was made that version 2 will be retired at the end of 2020 [3] therefore all users will migrate to version 3.

MicroPython is an implementation of Python that runs on microcontrollers that was developed in 2013 by Damien George and drops backward compatibility by focusing on Python 3.5. MicroPython [4] implements a subset of Python functionality and includes function and class libraries that are direct replacement for Python libraries but having as target the microcontroller market with their limitations: memory, hardware support and speed. For example the library cmath for mathematical functions for complex numbers is not available on the ESP8266 processor as it needs floating point support. Microcontrollers such as ESP32 or STM32 have floating point support and the functions in this library can be called [5]. As the multi core microcontrollers have started to appear for the low cost systems, the multithreading support in MicroPython is highly experimental and limited, while fully functional in Python.

Some variations of this software have appeared: CircuitPython in 2017 focused on simplicity and in the same year a version that runs on RISC-V processors [6] (that have seen an increase of use because they are open Instruction Set Architecture and offer trust and security).

The advantage of using MicroPython in the development of applications for microcontrollers is that there are many programmers that use Python as a programming language [7]. The transition to MicroPython is rather easy to do and involves mainly observing the limitations of the microcontrollers and optimizing the code in order to take advantage of its strengths and avoid the limitations (such as RAM speed and size, process of frequency and reduced number of cores).

Some of the disadvantages for developing MicroPython application include memory fragmentation and objects that grow in size (for example lists) [8].

This paper investigates the performance of microcontrollers running C and MicroPython for the ESP32 and STM32 devices in the following algorithms: computing Cyclic Redundancy Check (CRC) - used for error detection in data flows; floating point operations that involve memory allocation and management; computing SHA-256 (used in authentication, encryption algorithms and even cryptocurrencies).

II RELATED LITERATURE

MicroPython performance was investigated before in many articles because the hardware is cheap and the performance is at a level where it can process data locally instead of simply sending it to a server.

In the article [10] investigates the usability of the two main solutions for running Python on microcontrollers: MicroPython and CircuitPython. MicroPython was the first of the two to appear and is targeted towards obtaining the most performance from the hardware by scarifying the ease of use of the software. CircuitPython is a fork MicroPython and is promoted by Adafruit Industries and many code examples are linked to the hardware products sold by this firm. CircuitPython focuses on the ease of use of by offering many libraries that directly perform the required tasks without need for much understanding of the microcontrollers where the code runs, targeting the beginner programmers.

In the book [11] the hardware focus is on the boards: PyBoard, micro:bit, Circuit Playgroud Express and ESP8266 and MicroPython. The investigation is thorough presenting many of the pitfalls that decrease performance (especially the memory related errors that a Python user will not encounter) but many advances have happened in hardware especially since ESP32 has appeared (late 2016).

The book [12] investigates the use of both Arduino C and MicroPyhton for neural network development with one hidden layer on the ESP32 board. The research aims to investigate neural networks data propagation speed.

At the conference PyCon held at Portland in 2017, Jake Edge [13] has presented a comprehensive investigation in the causes of performance problems due to the memory management specific to Python and the limitations of the MicroPython environment. CPython was focused on simplicity of use (while boasting a multitude of extensions) while MicroPython has performance in mind (for example losing backward Python version compatibility). The conclusion was to avoid the corner cases where MicroPython performs poorly (such as lists and the garbage collection mechanisms) in order to keep the performance level good.

Finally in [14] present the known general issues with MicroPython performance and give hints about program optimization aspects that anyone writing code with these libraries should follow. However the paper does not target any specific processor and the two main aspects for code optimization: using arrays instead of lists due to their fixed nature and paying attention with memory allocation as there will be a significant slow down if garbage collection is activated. Other articles like [15] give insights on how MicroPython code works in byte code, native code, and native code with native types and how it can further be improved to increase execution speed (such as calling for integer addition instead of the C function rt_binary_op, the machine instruction adds, which gives a significant performance boost).

III Used hardware

Two of the most used microcontrollers for the low cost projects are STM32 and ESP32. Both are Arduino compatible, being capable of using a large software library.

ESP32 is a 40 nm process microcontroller based on the Xtensa 32 bit processor, designed by Espressif (a company that had a rapid evolution surpassing in 2018 100 million IoT chips [16]). ESP32 has become an important platform for developing low cost projects due to the integration of many connectivity solutions at a very low price point, with applications ranging from simple sensor reading, and WiFi/Blutooth communication, to video streaming, machine learning and facial recognition systems [17]. ESP32 has a performance of up to 600 DMIPS (32bit, at the top speed of 240MHz) on Dhrystone 2.1 benchmark [18]. At a cost of 2$ for individual boards it is a prime dual core solution. The disadvantage is having less documentation and support when compared to STM32 devices. The board used for tests is ESP-WROOM-32.

STM32 is a 32-bit ARM Cortex microcontroller that has a very low price and is integrated in many boards used in low cost projects. The 90nm device (an older technology when compared to ESP32) used for this paper is STM32F401RE [20], with a frequency of 84 MHz, 105 DMIPS and was part of the first STM32 series to have DSP instructions. The unit also presents a CRC calculation unit, unlike ESP32 that has to implement it solution via software. Therefore in computing power we should expect 6 times lower when compared to ESP32 when comparing purely computing performance. As a comparison, a Raspberry Pi 3B+ has a 3039.87 DMIPS performance [21]. The cost is the same with ESP32. It is a very well documented microcontroller and the libraries available target many application domains. Many of the microcontrollers in this series are pin by pin compatible, making easy to upgrade a project if more processing power is necessary. The first version of MicroPython was developed [19] on STMicroelectronics' STM32F4 board called pyboard. This is why in this paper a board with microcontroller from this family will be used: STM32 Nucleo.

The C development environment used was ESP-IDF for ESP32 and Visual Studio with VisualGDB for STM32. For MicroPython development uPyCraft was used for both platforms.

IV Implementation and Results

The algorithms tested are: floating point operation performance; CRC computation; Hash computation. The tests we repeated multiple times for each device in order to avoid measurement errors and to investigate result consistency.

The implementation of these algorithms was simple to create and test in MicroPython because the code was the same.

The C implementation however was different for the two platforms because of the function that measures the execution time and other platform particularities. For ESP32 the used function was *esp_timer_get_time*() while for STM32 the *coreticks* [22] function was used that measures time based on the processor clock frequency and from that execution time was computed. The STM32 measurement method works for all ARM chips that support DWT, and the Cortex M4 has this support. For MicroPython, the function used to measure time was time.ticks_us().

The first algorithm implemented was floating point operation performance, necessary in areas that require a larger range for results with the trade off being the precision loss (truncation, roundoff, etc.). This is a known weakness of the MicroPython implementation mainly because of the garbage collection operation. The implementation consists of a 300 000 for loop that performs an addition operation between floating point numbers. The results are shown in Table I and in Fig. 1.

TABLE I FLOATING POINT MEMORY ALLOCATION AND COMPUTATION TIME IN MICROPYTHON AND C FOR ESP32 AND STM32

	ESP32 Micro Python (µs)	STM32 Micro Python (µs)	ESP32 C (µs)	STM32 C (µs)
Run 1	1021756	2408869	6258	127923.06
Run 2	1024445	2404664	6257	127923.14
Run 3	1024419	2404196	6257	127923.06
Run 4	1021713	2402580	6258	127923.06
Run 5	1021723	2404892	6258	127923.06

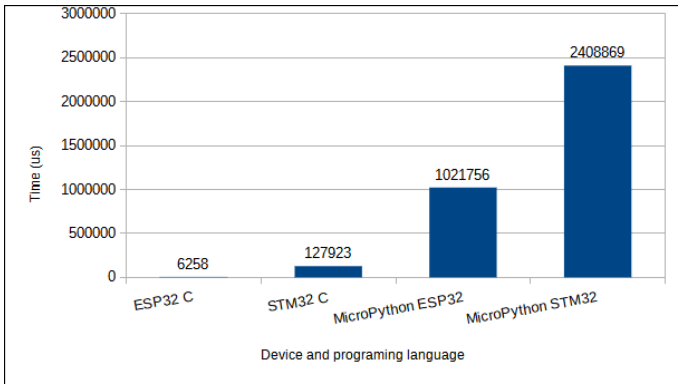

Fig. 1. Floating point computation results

As expected, the MicroPython is slower with an order of magnitude compared to the C implementation and ESP32 is faster than STM32 due to the clock and technology difference. The results also show that floating point operation is slower in MicroPython and should be avoided or replaced whenever possible with integer operations. As an additional observation, MicroPython has no problem with the allocation of a large number of floating point objects, as – for example – the use of: a = a + 0.1 in a loop will have to allocate the space for the object for every operation, instead of: a = a + v, where v is a constant value. This was noted as being a problem (double the time) in early versions of MicroPython [24]. This means that acceptable code can be written without paying attention to small details like frequency of variable allocation.

The second implementation was made for the CRC-32 algorithm computed for a 10 byte input. We have used the hardware CRC module for STM32 in C (offered by the TM_CRC_Calculate32 function) and the table implementation (same code) for ESP32 and in MicroPython. During research several aspects were observed: on Cortex M4 the CRC polynomial cannot be changed (Ethernet CRC-32 computation, while in higher modules like Cortex F7 the polynomial can be changed); the CRC hardware module in ESP32 is only used for calculating CRC of RTC fast memory; the C functions in ROM offered by ESP IDF use lookup tables to do computations in software. The results are presented in Table II and Fig. 2.

TABLE II CRC-32 COMPUTATION TIME IN MICROPYTHON AND C FOR ESP32 AND STM32

	ESP32 Micro Python (µs)	STM32 Micro Python (µs)	ESP32 C (µs)	STM32 C (µs)
Run 1	23056	15941	47	5
Run 2	23153	15952	47	5
Run 3	22780	16119	47	5
Run 4	23150	18961	47	6
Run 5	23602	19551	47	5

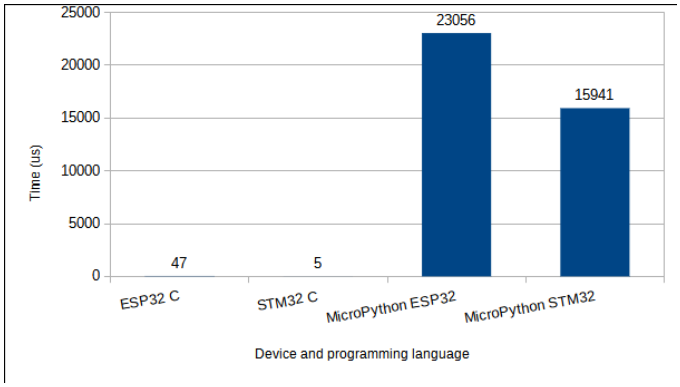

Fig. 2. CRC-32 computation results

The results show again that the C implementation is more efficient, in this case by three orders of magnitude. The hardware implementation used in the STM32 code allowed it to finish ahead of ESP32 even if the processor frequency favors the ESP32. However if other types of CRC need to be computed, the STM32 will perform worse than ESP32 as it will have to implement similar in memory lookup tables.

The final implementation is the SHA-256 hashing algorithm. The results are especially interesting to see because these micro platforms can be used to offload hash computation for blockchain transactions. The STM32F401RE does not have a hardware SHA-256 computation unit, this being available only in more expensive devices, like STM32F415. The implementation in this case was done in software, based on the algorithm implementation in [18]. ESP32 has hardware acceleration support for SHA and other cryptographic functions. It was easy to test the implementation using MicroPython using the hashlib library. The results for the ESP32 implementation are presented in Table III and Fig. 3.

TABLE III SHA-256 COMPUTATION TIME IN MICROPYTHON AND C FOR ESP32

	ESP32 Micro Python (µs)	ESP32 C (µs)
Run 1	295	163
Run 2	246	163
Run 3	244	163
Run 4	245	163
Run 5	245	163

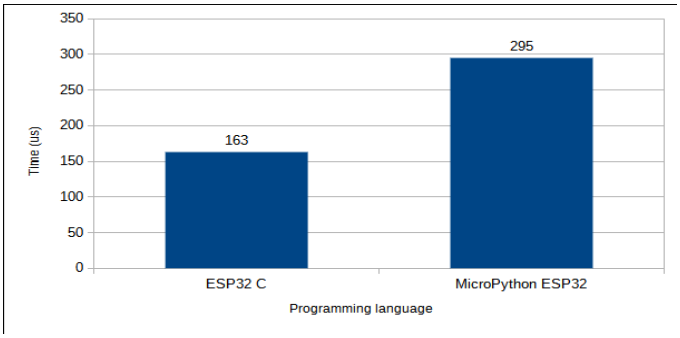

Fig. 3. SHA-256 computation results for ESP32

The C implementation is faster than the MicroPython one however only by 45%. Using hardware implementation for cryptographic functions gives better results then software implementations, however vulnerabilities found for the module in ESP32 [27] are harder to fix then a software implementation. The final observation will be made regarding the result consistency. The standard deviation of the results for of the MicroPython implementation is larger when compared to C, due to the memory operations that MicroPython is performing. Also, the standard deviation of the results for ESP32 is smaller when compared to STM32, in this case the motive is the much faster memory operations and processor speed in ESP32 that make memory management operations faster.

V CONCLUSIONS

In this paper the performance of MicroPython and C was tested for both STM32 and ESP32. The results show that the MicroPython performance level is lower than the C implementation as expected, but the difference varies depending of the task complexity and memory allocation. Writing code for MicroPython and comparing implementation results is simple because the code is portable, unlike C code where many platform specific changes needed to be performed. The result consistency is good only for the C execution. MicroPython shows large variations especially on the first runs. One of the possible applications of this study is teaching IoT programming [23] on low cost hardware using Python instead of C because of the code portability and ease of learning by students. In the future more test will be performed concerning network performance for MicroPython and C.

REFERENCES

[1] R. Olson, "Python usage survey 2014", January 30, 2015, web, http://www.randalolson.com/2015/01/30/python-usage-survey-2014/, Accessed 05.05.2020

[2] Jetbrains, "Python 3 vs Python 2", web, https://www.jetbrains.com/research/python-developers-survey-2017/, Accessed 05.05.2020

[3] Python Software Foundation, "Press Release 20-Dec-2019, PYTHON 2 SERIES TO BE RETIRED BY APRIL 2020", web, https://www.python.org/psf/press-release/pr20191220/, Accessed 05.05.2020

[4] D.P. George et al., "MicroPython Documentation Release 1.10 Jan 25, 2019",web, http://docs.micropython.org/en/v1.10/micropython-docs.pdf

[5] A. Schweizer, "ESP32 floating-point performance", 2016, web, https://blog.classycode.com/esp32-floating-point-performance-6e9f6f567a69, Accessed 05.05.2020

[6] B. Nilawar, "MicroPython Port for RISC-V soft Processor", 7th RISC-V Workshop Proceedings, November 28 – November 30, 2017, California

[7] L. Tung "Programming languages: Python developers now outnumber Java ones", 2019, web, https://www.zdnet.com/article/programming-languages-python-developers-now-outnumber-java-ones/, Accessed 05.05.2020

[8] D.P. George et al., "MicroPython on microcontrollers", 28 Jun 2020, web, https://docs.micropython.org/en/latest/reference/constrained.html, Accessed 05.05.2020

[9] J. Ivković, J. Ivković, "Analysis of the performance of the new generation of 32-bit Microcontrollers for IoT and Big Data Application", Proceedings of ICIST 2017, p330-336.

[10] -, "Profile: Scott shawcroft: This developer is squeezing python into microcontrollers," in IEEE Spectrum, vol. 56, no. 4, pp. 16-16, April 2019, doi: 10.1109/MSPEC.2019.8678507

[11] N.H. Tollervey, "Programming with MicroPython: Embedded Programming with Microcontrollers and Python", O'Reilly Media, Inc., Sep 25, 2017, ISBN 978=1=491-97273-1

[12] Dokic K., Radisic B., Cobovic M. "MicroPython or Arduino C for ESP32 - Efficiency for Neural Network Edge Devices", In: Brito-Loeza C., Espinosa-Romero A., Martin-Gonzalez A., Safi A. (eds) Intelligent Computing Systems. ISICS 2020. Communications in Computer and Information Science, vol 1187. Springer, Cham

[13] J. Edge, "Memory use in CPython and MicroPython", web, 2017, https://lwn.net/Articles/725508/, Accessed 05.05.2020

[14] D.P. George, P. Sokolovsky, "Maximising MicroPython speed", web, https://docs.micropython.org/en/latest/reference/speed_python.html, Accessed 05.05.202

[15] D. George, "The 3 different code emitters", 2013, web, https://www.kickstarter.com/projects/214379695/micro-python-python-for-microcontrollers/posts/664832, Accessed 05.05.202

[16] Espressif, "Espressif Achieves the 100-Million Target for IoT Chip Shipments", web, https://www.espressif.com/en/news/Espressif_Achieves_the_Hundredmillion_Target_for_IoT_Chip_Shipments, Accessed 05.05.2020

[17] M. Gravråk, "Machine Learning at the Edge with ESP32", web, https://www.bouvet.no/bouvet-deler/machine-learning-with-esp32, Accessed 05.05.2020

[18] B. Conte, "sha256.c", web, https://raw.githubusercontent.com/B-Con/crypto-algorithms/master/sha256.c, Accessed 05.05.2020

[19] J. Beningo, "Prototype to production: MicroPython under the hood". EDN Network. 2016, web, https://www.edn.com/prototype-to-production-micropython-under-the-hood/,

[20] STMicroelectronics, "STM32F401RE Datasheet", web: https://www.st.com/resource/en/datasheet/stm32f401re.pdf, Accessed 05.05.2020

[21] R. Longbottom, "Raspberry Pi 4B 64 Bit Benchmarks and Stress Tests". 10.13140/RG.2.2.33099.54562.

[22] Erich Styger, Cycle Counting on ARM Cortex-M with DWT, January 30, 2017, web, https://mcuoneclipse.com/2017/01/30/cycle-counting-on-arm-cortex-m-with-dwt/, Accessed 05.05.2020

[23] Pearson, Bryan & Luo, Lan & Zou, Cliff & Crain, Jacob & Jin, Yier and Fu, Xinwen. (2020). Building a Low-Cost and State-of-the-Art IoT Security Hands-On Laboratory. 10.1007/978-3-030-43605-6_17.

[24] Github, ""gc speedup" changes led to much increased heap usage #836", web, https://github.com/micropython/micropython/issues/836, Accessed 05.05.2020

[25] STMicroelectronics, "RM0368 Reference manual", web, https://www.st.com/resource/en/reference_manual/dm00096844-stm32f401xb-c-and-stm32f401xd-e-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf, Accessed 05.05.2020

[26] Limited Results, "Pwn the ESP32 crypto-core", August 2019 web, https://limitedresults.com/2019/08/pwn-the-esp32-crypto-core/, Accessed 05.05.2020