Arduino Profiling: An execution time and CPU availability investigation

Júlia Moura [a].

[a] Engineering School, Federal University of Minas Gerais, Minas Gerais, Brazil, julia.santos.moura.1201@gmail.com

Abstract. As the world gets more connected the importance of IoT grows. A common microcontroller used as an Iot device is Arduino due its great flexibility, small cost, and size. Although there are studies about this microcontroller, they do not deepen its execution time and CPU availability profiling, which is the main goal of this article. To do that two different methodologies were chosen to measure how long it took to execute 8 different functions, in which four of them had three different data sizes. The results of both methodologies were consistent and proved the good performance of Arduino. To measure the CPU availability a library was used, but its result was not conclusive.

Keywords. Profiling, Measure Performance, Tracing performance analysis, Arduino, code, software, library, framework.

1. Introduction

Many applications built on embedded platforms are cheap, small, and with flexible computer hardware. Mostly, such platforms are based on microcontrollers (MCUs), i.e., single-chip (micro)computers that have a goal to govern a specific operation in an embedded system. One of the most popular MCU-based platforms is Arduino we build on in this paper.

Especially in the area of more complex and critical applications, there is a need to identify key parameters of embedded systems to quantify parameters such as interrupt latency and response time, best/worst-case execution time, CPU/stack utilization, time spent in various operating modes (e.g., run, wait, stop, interrupt handling) or power consumption. Such an analysis can be done, e.g., by means of the so-called profiling.

Some of Arduino's characteristics such as clock speed, communication parameters supported and I/O Connectivity have been analyzed in Swathi's work [1]; another work, by Suresh [2] investigates more deeply how is the execution time when an MCU executes different functions. Another useful information that can be profiled for Arduino is CPU availability since it can be useful to identify the consumed CPU time.

There are many ways to profile parameters like that, as explained by Patel [3]. It could have been done with dedicated hardware, but the chosen approach was the software one, following the example of Strasser [4] and avoiding software overhead.

2. Research Methods

2.1 Execution Time Profiling

To measure the execution time it was important to define a set of events that would be profiled. Those events needed to be varied in complexity to reflect the different times needed by Arduino to perform those functions. It was also important, as highlighted by Suresh [2], to make the measurements with different amounts of data.

Suresh [2] also influenced deeply on the chosen functions, which are shown above. But, Strasser's [4] work also had a great influence on it:

- Initialize variable (1)
- Invoke an empty function (2)
- Acquire and release a fixed amount of memory (3)
- Create a vector of words (4)
- Perform string search on a vector of words (5)
- Execute three basic math functions: square root, cubic and degree to radian conversion (6)
- Sort data using qSort function (7)
- Search for the smaller path in a graph using Dijkstra algorithm. (8)

Again, guided by Suresh's [4] methodology, the functions 4, 5, 7, and 8 were measured using different sizes of data, divided into three categories: small (S) with 10 units, medium (M) with 50 units, and large (L) with a 100.

The data size was restricted like that due to memory issues presented by Arduino when vectors of strings with sizes bigger than one hundred were created.

As shown in Strasser's [4] thesis, the main idea was to measure the time before and after the execution of the event and then subtract the found values to find the execution time.

The first methodology was made with the *micros* function. And, to avoid initialization overhead and latency the measure was made before and after a loop in which the given event was executed. Then, the found result was divided by the number of times the loop was executed (1000 times).

In the second method, the measurement was made with the Time Profiler Library. Using the *TIMEPROFILE_BEGIN*, *TIMEPROFILE_END,* and *TimeProfiler.getProfile* functions. To obtain more reliable results the profiling was made on the loop function from Arduino.

In both approaches, an Arduino Uno R3 was used with a cable USB-A/ USB-B and an Aspire A515-55 computer. The materials are shown in more detail in **Tab. 1**.

Tab. 1 - Materials used in the experiments.

material	detail	image
ARDUINO UNO R3	Microcontroller: Atmega328P Minimum voltage: 5 V Maximum voltage: 12 V Operating voltage: 5V	
ASPIRE A515-55	Processor: Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz 1.19 GHz RAM: 8,00 GB (usable: 7,78 GB) System type: operational system of 64 bits, processor based in x64	
Cable	USB-A/USB-B 53 cm	

All the codes were implemented on Arduino IDE version 2.0.4 for Windows 64 bits.

2.2 CPU Availability Profiling

Another aspect of Arduino that was analyzed is CPU availability, which says where the time is spent on the CPU. Normally, a development environment uses a profiling hardware to sample the program counter of the investigated hardware on a regular basis and counts when it lands in a range of addresses (bins) previously determined.

But, this is not possible on Arduino since there is not enough RAM to divide the program space into many bins and have a dedicated counter for each one of them. To solve that issue Dudley [5] developed a library that manually defines a bin for samples of code. This library was used to profile the cpu availability of the functions mentioned before.

The sample code from Dudley [5] prints between the 3rd and 4th seconds each of the defined bins and the number of times they were hit during the 1-minute sample time. His code was modified to invoke each one of the events profiled by this article at bin number seven.

3. Discussion of Results

3.1 Execution Time

In **Tab. 2** all the profiled events and their correspondent execution time obtained with the two methodologies explained before are shown.

The first methodology (A), is the one in which the *micros* function is invoked before and after a loop that executes the event and the final time is calculated. The second method (B), is the one in which the execution time was profiled using the TimeProfiler library.

Tab. 2 - Results obtained of the time of execution for each profiled event with the two methodologies.

function	time A (ms)	time B (ms)
Initialize variable (1)	4	0.012
Invoke an empty function (2)	0	0.012
Acquire and release a fixed amount of memory (3)	0	0.012
Create a vector of words (4)	S: 1 M: 9 L: 9	S: 1.024 M: 5.748 L: 9
Perform string	S: 0	S: 0.012

search on a vector of words (5)	M: 0 L: 0	M: 0.012 L: 0.008
Execute three basic math functions: square root, cubic and degree to radian conversion (6)	0	0.012
Sort data using qSort function (7)	S: 0 M: 0 L: 0	S: 0.112 M: 0.448 L: 0.820
Search for the smaller path in a graph using Dijkstra algorithm. (8)	S: 0 M: 0 L: 0	S: 0.012 M: 0.012 L: 0.012

Based on the results from the functions 2, 3, 5, 6, 7, and 8 can be notice that the microseconds scale is not precise enough to measure the events execution time, since to all of those functions the result was 0. It indicates that the execution time is on the scale of nanoseconds, but Arduino does not provide a function to measure it.

That is the main reason why the second approach was chosen, since, even though it is also on the microseconds scale it is more precise because it provides the decimal parts of the measure.

With that, it is possible to validate that the results from functions 2, 3, 5, 6, 7, and 8 are correct. Since the obtained values for those functions with methodology B are always smaller than one microsecond. So, for those cases, both methodologies provided similar results.

It is important to highlight that as approach B is more precise it is noticeable that in function 7 the execution time varied according to the sample size, but it was always smaller than 1 microsecond. That level of analysis is not possible with methodology A.

Some results diverged between the two methodologies. In function 1, the result of the first approach is weird, since it takes 4 microseconds to just create a variable. But, in practice, this event is extremely similar to function 3, in which an amount of memory is acquired.

The result for the same function, 1, with methodology B is more consistent. Since it takes 0.012 microseconds, as occurs in function 3.

Another measurement divergency is from function 4 with size M, in which a vector of 50 words is created. With methodology A it took 9 microseconds to run, while in methodology B the time was smaller: 5.478 microseconds. The reason for that happening was not identified.

This difference did not occur for the other sizes. For 10 words (S) in the first method, the time was 1 microsecond, and on the second one 1.024 microseconds, which emphasizes that the *micros* function is not precise enough. But for the largest sample, 100 words, the result was exactly the same: 9 microseconds.

3.2 CPU Availability

Moving to the second point of analysis: the CPU availability, we have on **Tab. 3** the amount of time the range of address of each function (bin) was hit during its execution:

Tab. 3 - Results obtained of the cpu availability for each function with the Dudley library [5].

function	number of hits on the bin
Initialize variable (1)	0
Invoke an empty function (2)	0
Acquire and release a fixed amount of memory (3)	0
Create a vector of words (4)	S: 65 M: 335 L: 611
Perform string search on a vector of words (5)	S: 0 M: 0 L: 0
Execute three basic math functions: square root, cubic and degree to radian conversion (6)	0
Sort data using qSort function (7)	S: 5 M: 17 L: 34
Search for the smaller path in a graph using Dijkstra algorithm. (8)	S: 0 M: 0 L: 0

The result was 0 for the following functions: 1, 2, 3, 5, 6, and 8. As the execution time of those functions is close to 0 it is possible that those functions execute so fast that the library was not able to profile them.

Even though function 7 also has an execution time small, it is closer to one microsecond when compared to the functions mentioned before. This time was long enough for the event to be profiled and is possible to notice the consistency in the results, which increases as the sample size grows.

The same thing happened to function 3 and is possible to notice the number of times the functions are hit grows almost linearly with their size:

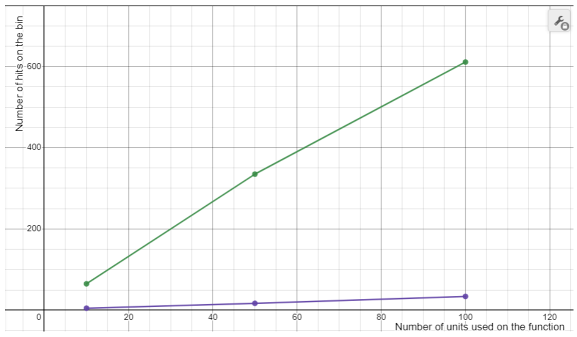

Fig. 1 - Size of the function and the number of times they are hit. In green function 4 and in purple function 7.

4. Conclusion

Arduino is a cheap, small, and flexible computer hardware that is able to connect with many sensors to collect different types of data. Because of that it is a great option to be used as an IoT device. That justifies the purpose of this work of profiling its execution time and CPU availability.

Two different methodologies were used to measure the time aspect and both of them offered consistent results. Even though the functions varied in complexity and size to most of them the execution time was smaller than 1 microsecond. This is a great indicator of good performance since the functions are executed pretty fast.

The measured CPU availability results were not conclusive because for the most part of the functions, due to their small execution time, they ran so fast that the Dudley library [5] could not measure how many times they hit the selected range of addresses from the functions.

4. Acknowledgments

The author would like to thank Ing. Josef Strnadel from the Institute of Czech-Brazilian Academic Cooperation for all his help in the profiling of Arduino aspects and in the writing of this paper.

5. References

[1] Swathi K. T., Sandeep T. U, Ramani A. R., Performance Analysis of Microcontrollers Used In Iot Technology. *International Journal of Scientific Research in Science, Engineering and Technology.* 2018; 4: 6.

[2] Suresh A. J., Siriram S.Power Profiling and Analysis of Code Obfuscation for Embedded Devices. *IEEE India Council International Conference (INDICON)*. 2020; 1: 6.

[3] Patel R, Rajawat A. A survey of embedded software profiling methodologies. *International Journal of Embedded Systems and Applications*. 2011; 2: 22.

[4] Strasser F. J., *Profiling of Real-Time Operating System Services for Singlecore and Multicore.* Graz University of Technology. Graz; 2014. 101.

[5] Dudley W. F., *Profiling Arduino Code.* William Dudley Projects. 2023. https://www.dudley.nu/arduino_profiling/.