Application of Modern Programming Languages in solving the Problem of Emulator Development for Embedded Systems

Andrey Vlasov\*[ 0000-0001-5581-4982], Aleksey Gladkikh\*[0009-0005-0416-4533]

and Kirill Kutaev\*[0009-0001-6212-1540]

Bauman Moscow State Technical University, 2-nd Baumanskaya 5, Moscow, 105005, Russia  
\*vlasov@iu4.ru

**Abstract.** The work is devoted to the use of modern programming languages in solving the problem of developing a hardware platform emulator used for debugging the developed software of embedded systems. The authors perform a research of the performance characteristics of modern programming languages in performing certain operations and algorithms most often used in the process of emulating embedded systems using the software developed as part of the research. The paper discusses the characteristics of existing emulators of built systems and offers recommendations for developing an emulator that has higher performance characteristics and more efficient support potential for it as a software product. The results obtained can be useful for engineers and developers working in the field of embedded systems design and software development for them. #CSOC1120

**Keywords:** Embedded System, Emulator, Programming Language, Processor, Microcontroller.

1. Introduction

In the modern world, embedded systems are an integral part of most branches of human activity [1]. An embedded computing system is any computing hardware and software system that is not a general-purpose system such as a personal computer or server. Their applications cover a wide variety of fields including automotive, medical, energy, security systems, industrial manufacturing, home appliances, etc. Internet of Things (IoT) technology also relies heavily on embedded systems to create a network of devices that can be remotely monitored and controlled [2].

In the process of developing software for embedded systems, there is often a need to debug and test it. It is not always possible to perform these operations on the embedded system under development due to lack of physical access to it or lack of necessary hardware parts on it that are still in the process of design or production. Even with a fully designed hardware part of the embedded system in some cases there is no possibility to get detailed information about all processes occurring in it, which must be debugged and tested. This is due to the fact that modern embedded systems use many hardware components, between which there is various communication, and it is difficult to monitor software states in all components at the same time [3]. These communication processes may also depend on external events, and then the complexity of debugging increases significantly [4].

Running the software of embedded systems on the developer's computing system (personal computer) can also be difficult. One of the reasons is the difference between processor architectures of the developer's computing system and the target system for which the development is carried out. Embedded systems often use specialized processors or microcontrollers that are oriented to perform highly specialized tasks with low power consumption [1]. One way to solve this problem is to use an emulator - a software tool that simulates (emulates) the operation of a computing system and its environment in another computing system, so that the simulated behavior corresponds as closely as possible to the behavior of the real computing system.

The work is dedicated to the application of modern programming languages to solve the problem of developing a hardware platform emulator used for debugging the developed software of embedded systems in conditions when debugging the system physically is difficult or impossible. Embedded computing systems have become widespread in recent decades as more and more processes in both everyday life and industry require a high degree of automation and integration [2]. Thus, the efficiency of software development for embedded systems becomes a critical aspect in their production.

The object of the research is modern tools and methods of software development, in particular imperative programming languages that were popular at the time of writing this article. Imperative programming language is a formal language used for writing computer programs that define the sequence of commands executed by the processor. The basic idea of imperative programming is to tell the computer system what operations to perform and in what order.

The subject of the research is the fast performance of modern imperative programming languages in the context of the emulation task of embedded systems software, as well as their capabilities for efficient support and development of new emulator components for embedded systems.

The purpose of the work is to investigate the characteristics of the speed performance of modern programming languages in the execution of certain operations and algorithms, most often used in the process of emulation of embedded systems. The result is a software tool to conduct a research of programming languages to determine the speed of execution of specified algorithms, as well as proposals for the use of software development tools to implement a modern emulator of embedded systems that meets the criteria of speed and efficiency of supporting it as a software product.

The relevance of the research is due to the development of information technology and the increasing integration of embedded systems in many areas of industry and everyday life. Embedded software becomes more and more complex and gets more and more extensive functionality. Because of this, the number of processes occurring in embedded systems software is also increasing. This leads to an exponential increase in the number of possible paths of software execution [5]. Testing, debugging, and analyzing the execution of such software becomes more difficult. With the increasing integration of embedded systems into a wide range of automation processes, efficient software development and testing become critical components of developing reliable, fault-tolerant, and secure embedded systems [6].

It is also important to mention that in industry, where embedded systems are widely used, small deviations in the expected behavior of devices can lead to serious consequences, including damage to critical infrastructure and harm to humans.

One of the possible ways to increase the efficiency of testing and debugging is to run software in an emulator. Emulators have a wide range of applications [7]. They are used, in particular, for dynamic analysis of software execution within the framework of certification tests [8], for performance testing [9], for confirmation of correct behavior of devices or programs [10], and for solving reverse engineering tasks [11].

The use of emulation as a method of software testing and debugging has existed for quite a long time, since the first computing systems [12]. It has a number of advantages over the use of physical hardware. When emulation is started, the software to be tested is loaded by writing it directly into the emulated memory. Thus, there is no need to load the software over a serial line or network and start monitoring the state of the target embedded system. The emulated embedded system is controlled over the same command transfer interface as the real physical system. Consequently, there is no need for physical access to the target embedded system.

Due to the fact that the emulator runs on a developer workstation, it is possible to create test scenarios and automate their launch. Since testing is performed by running a normal process (emulator) on the operating system of a workstation, test suites (multiple individual test cases) can be run simultaneously on multiple workstations. Emulation determinism simplifies regression testing, which is a check of a previously tested part of the software to make sure that the changes made did not cause defects in the part of the software that was not changed. Any change in performance results compared to the results of previous versions of the same software can only be caused by differences in the software under test and not by hardware failures or other random changes [13].

In general, the use of emulator of embedded systems with the characteristics of speed and efficiency of its support as a software product represents an important step in improving the quality of developed systems for both industrial and everyday applications.

1. Background and Related Works

Development of emulator of embedded systems is a non-standard task due to its low prevalence and the small number of existing solutions. When designing such software, the choice of programming language is an important task, because the characteristics of the developed software product and the effectiveness of its support will depend on this choice. Many programming languages are used for modern software development. According to the rating on the TIOBE index, about 50 programming languages currently occupy 70% of the market share [14]. Each of them has its own application area, philosophy, features, different dialects, frameworks and many other characteristics by which they can be compared.

Studies show that the right choice of tools and instruments in software development has a significant impact on the development process, the quality of the result and the ability to further support the product, especially for various critical systems [15]. Although most of the programming languages are universal in terms of producing the required result, some of them are more suitable for specific purposes than others [16]. In order to reduce the number of choices, when selecting a programming language, it is necessary to start from the goals that are achieved by implementing the given software. For this purpose, it is necessary to evaluate a set of criteria that the developed product should possess. Criteria such as speed of operation, memory efficiency, dependence or independence from the platform, availability of user interface, execution environment, etc. can be used.

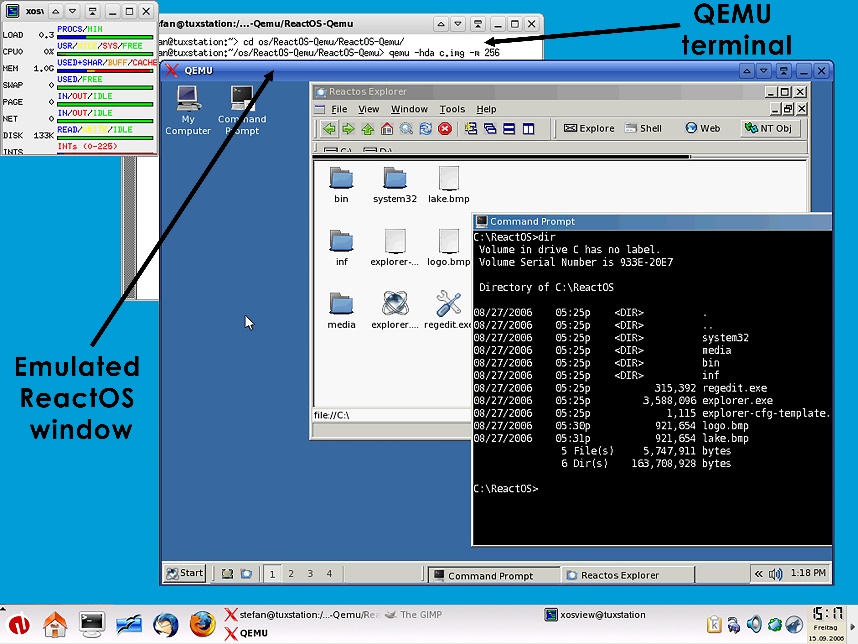
Since a programming language is an abstract description of a computational process, only the process itself and its results can be objectively measured. The essence of such methods of comparing programming languages is that the code written in different languages and executing the same process to get the same result is evaluated. It is necessary to take into account the difference between the programming language and the program when using this method and interpreting the results [16]. When comparing programming languages, one should use only basic syntactic constructions of the language to achieve the most objective performance evaluation. This is due to the presence of unique syntactic abstractions, the principle of operation of which in the internal structure of the language is not obvious or unknown and thus cannot be accurately reproduced in other programming languages.

Thus, the choice of a programming language for writing modern software is a complicated task that requires a complex approach and the use of various methods of analysis. However, the development of methods and software tools for comparing the performance of programming languages is an important direction for the design of modern software that meets the criteria of speed and efficiency of its support as a software product.

When designing a modern emulator of embedded systems, it is necessary to identify the criteria it should meet. Below is a review and analysis of existing solutions and analogs of emulator of embedded systems.

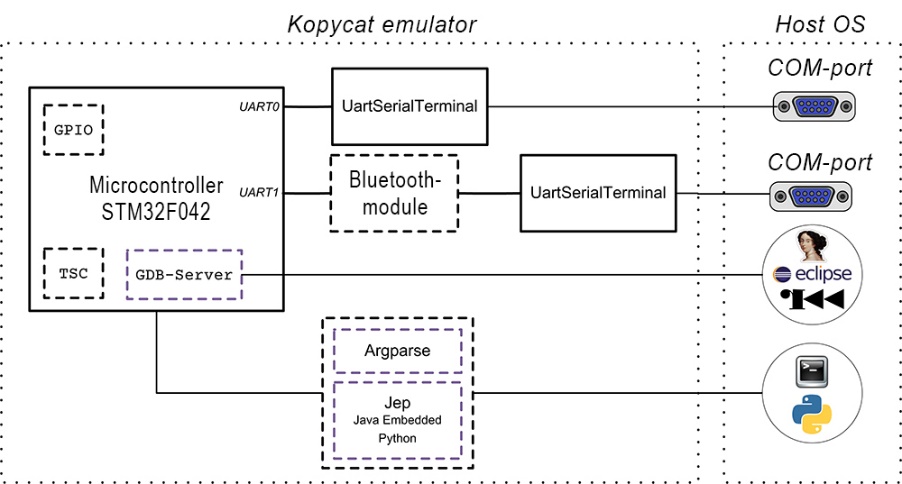
**Qemu** (Quick Emulator) is an open-source tool that is used to emulate various architectures [17]. Qemu has a large developer community and as a consequence, good support. It supports two emulation modes: user mode, in which a complete emulation of the system, including processor and peripherals, takes place, and system mode, in which only translation of instructions and system calls of the emulated system takes place. Figure 1 shows an example of Qemu in system mode, where the operating system ReactOS is emulated.

Qemu has many positive features like RSP GDB (Remote Serial Protocol GNU Debugger) interface, modeling based on logical bus coupling and high performance, since it is written in C language. On the other hand, it creates a big disadvantage in the form of high complexity of support and extension, in particular writing new modules (architectures, peripherals, etc.).



**Fig. 1.** Emulation of the ReactOS operating system in the Qemu emulator.

**Kopycat** is a Russian emulator of hardware platforms that allows low-level program emulation of arbitrary hardware systems and their debugging through the GDB interface supported by most development tools [18]. It is implemented in the Kotlin (Java) programming language, the main advantage of which is the efficiency of developing new solutions. Figure 2 shows the emulator architecture for emulating the STM32 series microcontroller.

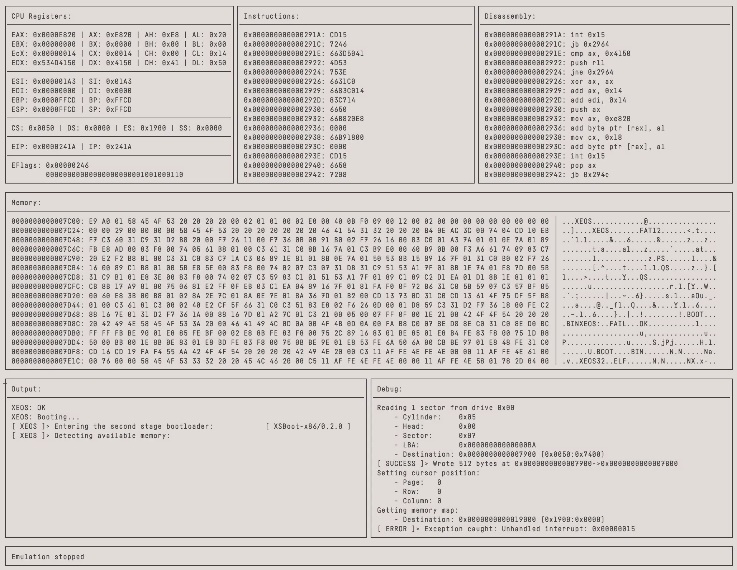


**Fig. 2.** Architecture of Kopycat emulator for emulation of STM32 series microcontroller.

One of the competitive advantages of Kopycat is the availability of a convenient software development kit and documentation for creating new computing cores and peripherals [19]. There are two possible approaches for creating new modules: the first is a structural approach, the second is a behavioral description. During structural description, the installed modules and connections between them are specified in a file in JSON (JavaScript Object Notation) format. Structural description can also be performed using the Kotlin/Java programming language. In behavioral description, handlers for writing or reading from certain addresses can be added to the module. Inside the handler it is specified how the module should handle this event. Behavioral description can be specified only in the Kotlin/Java programming language. Among the disadvantages, we should note the average performance when emulating embedded systems that do not have full-fledged general-purpose operating systems (for example, distributions based on the Linux kernel). In case of emulation of such systems, performance is significantly degraded.

**Unicorn** is a lightweight, multi-platform and multi-architectural processor emulator. This emulator differs in its architecture from the others in the list by having only basic functionality [20]. It does not emulate the operation of all software or an entire operating system.

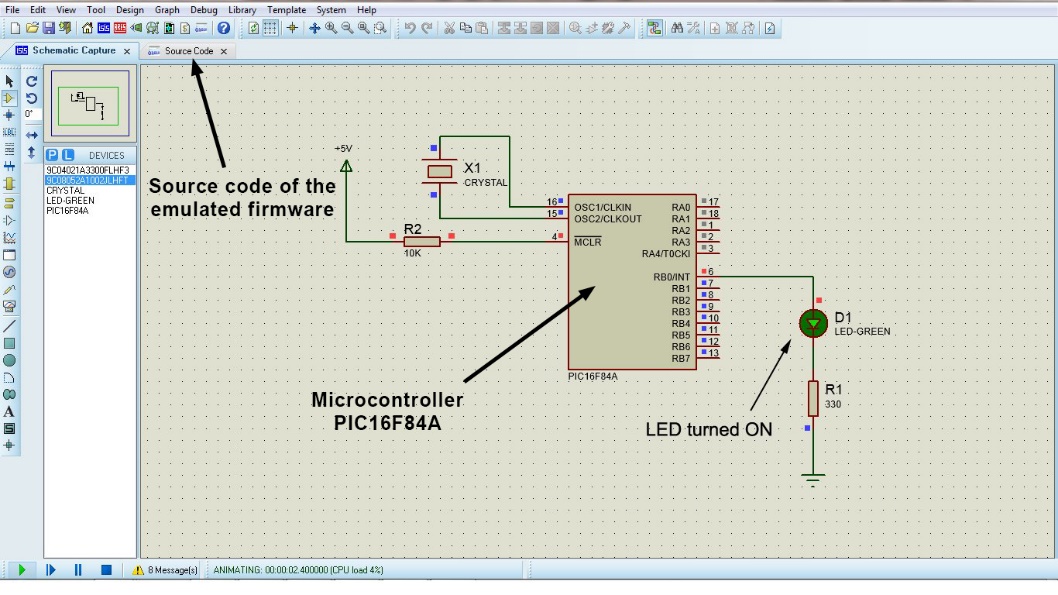
It does not support system commands (such as opening a file, outputting a character to the console, etc.). The developer has to partition the memory of the emulated software and load data into it himself. Figure 3 shows the graphical interface of BIOS (Basic Input/Output System) emulation in Unicorn.



**Fig. 3.** BIOS emulation in Unicorn.

The disadvantages of this emulator include the lack of built-in abstractions and tools for software operation, configuration and use can be difficult for inexperienced users, the lack of system commands, the complexity of implementing new modules of the emulator. Thus, working with Unicorn requires high qualification of a specialist.

**Proteus** is a software package for computer-aided design of electronic circuits, the distinctive feature of which is the ability to simulate the operation of programmable devices [21]. Emulation of microcontroller of PIC16 family is presented in Fig. 4. Emulation in Proteus, unlike Qemu, takes place at the level of electrical signals, which significantly slows down its speed.



**Fig. 4.** Emulation of the PIC16 family microcontroller in Proteus.

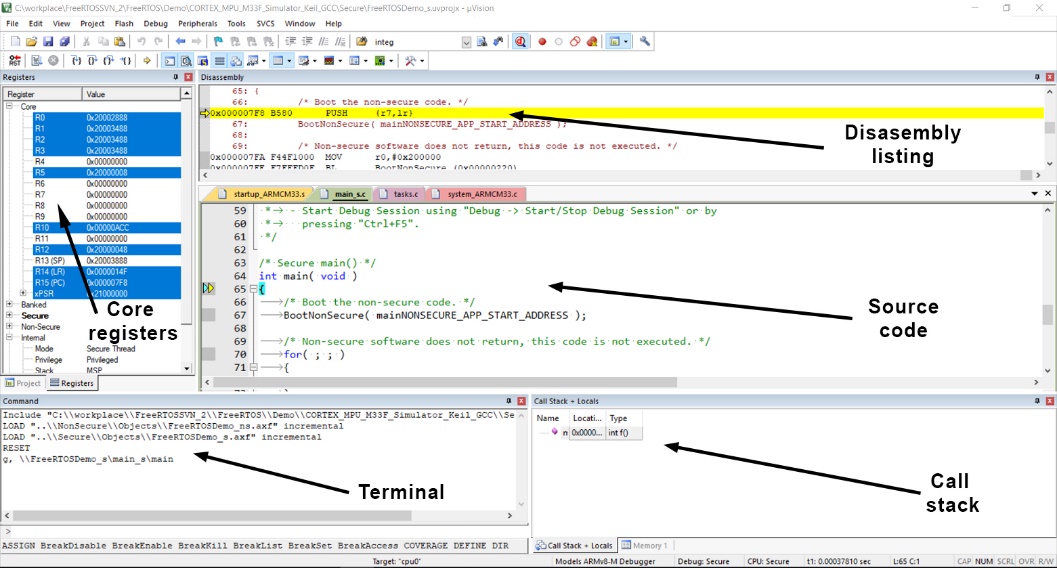
Proteus is proprietary software, unlike the emulators described above, but there is a possibility to write your own modules in C++. Their implementation requires high qualification of a specialist due to the lack of detailed documentation, weak support in the developer community and peculiarities of C++ programming language.

**Emulators embedded in** **IDE** - some IDE (Integrated Development Environment), for example, ARM Keil, have a built-in emulator of hardware platforms (Fig. 5) [22]. Such emulators have poor support of the developer community, low performance and are practically inapplicable in the tasks of embedded systems development.

This is caused by the absence of the possibility to implement their own modules and processor architectures, as well as to emulate peripherals, which are an obligatory component of any embedded system.

The comparison of existing solutions and emulator analogs of embedded systems is presented in Table 1. The values of qualitative characteristics are given relative to the elements of this table.

As a result of consideration of this issue, we can conclude that existing solutions for emulation of embedded systems do not have the characteristic of speed and the ability to easily implement new modules at the same time. At high speed of the C programming language, they have a small number of built-in tools and abstractions and are poorly developed.



**Fig. 5.** Emulation of ARM Cortex-M33 core operation in Keil uVision IDE.

**Table 1.** Comparison of embedded system emulation and circuit simulation tools.

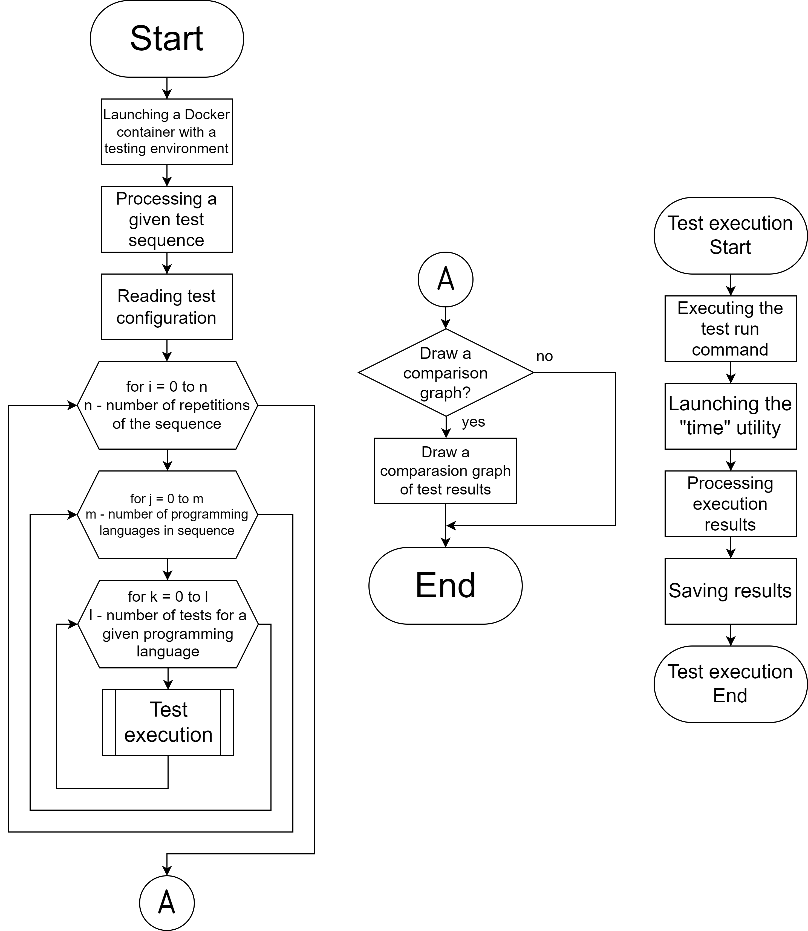
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Software | Performance | Difficulty in implementing new modules | GDB interface | Modeling of electrical signals |
| Qemu | High | High | + | – |
| Kopycat | Average | Low | + | – |
| Unicorn | High | High | + | – |
| Proteus | Low | High | – | + |
| Keil IDE | Low | None | + | – |

On the other hand, C++ programming language has a large number of built-in facilities and abstractions, but the complexity of development in it is extremely high and the resulting outputs often have non-intuitive behavior. Thus, there is a need to develop an emulator that will have high performance characteristics and ease of implementation of new modules.

1. Materials and Methods

The key method of the research is to test the speed of execution of given algorithms by different programming languages and then compare the results of the experiments.

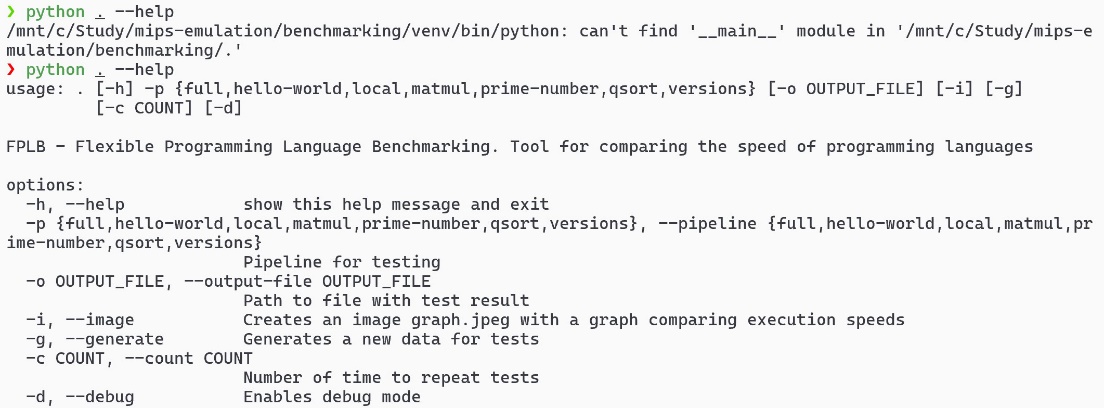
In order to test the performance of programming languages, we developed "FPLB" (Flexible Programming Language Benchmarking) [23] with the necessary environment for testing, in the design of which we used the research presented in the work "Cross-Language Compiler Benchmarking" [15]. The flowchart of the algorithm of the "FPLB" software is presented in Fig. 6.



**Fig. 6.** The flowchart of the algorithm of the "FPLB" software operation.

The "FPLB" software allows to run various pre-prepared algorithms for specified programming languages in an automated mode. It also allows to log test results, draw a bar graph based on test results, generate the necessary input data for testing, as well as run a sequence of identical tests and calculate the average value of the test results. The output of the reference data on the use of the "FPLB" software is shown in Fig. 7.

The Python programming language was chosen for the development of the FPLB software because it is easy to develop, debug, and maintain software in. It has a large and active community of developers, which provides many resources, documentation, libraries and development tools. The Docker container platform was chosen as the test execution environment. It is a common lightweight and self-contained runtime container that includes all the necessary environment to run the software, including libraries, system tools, and runtime environment. It keeps the environment of the algorithms under test in the same stable state, which reduces the impact of different processes running on the computer system under test.



**Fig. 7.** Output of reference data on the use of the "FPLB" software.

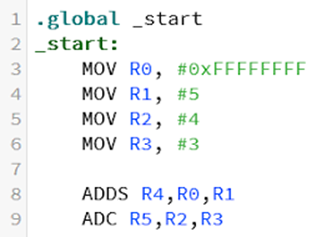
To estimate the execution speed of the tested software, we used the method proposed in the research "Comparative Testing of Programming Languages" [16]. The value of the execution time of the tested software in a certain programming language is measured using the "time" software. This is a standard UNIX utility that outputs a statistical message about the time spent to execute the software passed to the utility as input.

The choice of algorithms, the speed of execution of which will be tested, was based on the work " Research of Speeds of Performance of Basic Mathematical Problems of Popular Programming Languages" [24]. Since the purpose of the work is to research the characteristic of programming languages' performance in the execution of algorithms most often used in the emulation of embedded systems, it is necessary to choose those algorithms that contain the operations most often used in the emulator's work. These operations are reading and writing data to memory, as well as basic arithmetic operations and comparison operations. This is due to the specificity of modern processor architectures [25]. Thus, algorithms containing a set of such operations are required.

Visualization of the results was performed using the method used in the research "Cross-Language Compiler Benchmarking" [15]. For this purpose, the library for the Python language "MatPlotLib" was used. It allows visualization of data by two-dimensional and three-dimensional graphs. Thus, it is possible to visually evaluate the ratio of algorithm execution speeds of different programming languages.

* 1. Analysis of Software Execution

From the processor's point of view, software is a sequence of instructions describing the actions to be performed on a set of data. These processor instructions can be written as mnemonic codes in Assembly language. All Assembly languages are machine-specific, that is, they are designed for a particular type of processor architecture. Examples of mnemonics in Assembly language for the ARM (Advanced RISC Machine) architecture are shown in Fig. 8.



**Fig. 8.** Instruction mnemonics in ARM assembly language.

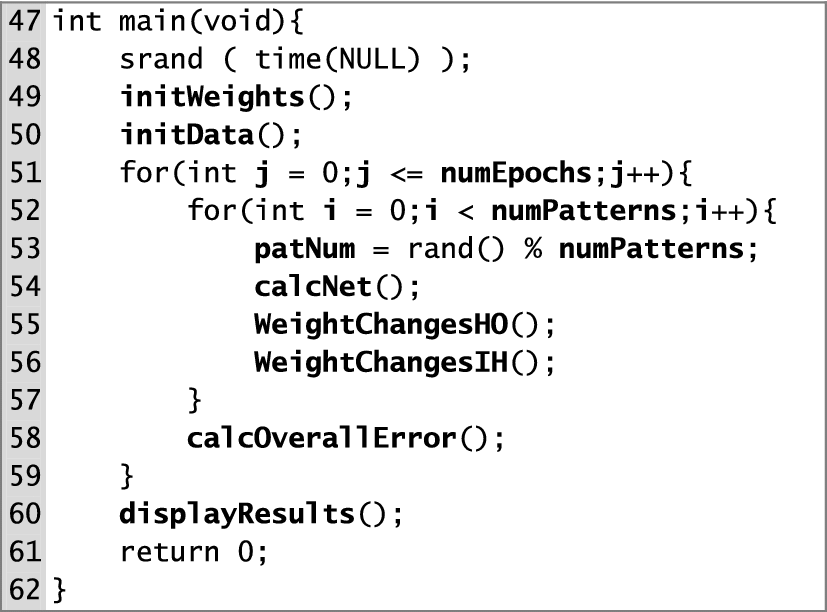
From a software developer's point of view, it is a syntactic unit that follows the rules of a particular programming language, consisting of definitions, operators or instructions necessary for a particular function, task or problem solving. It is represented as a text file with a set of source code. An example of C source code is shown in Fig. 9.

In order for the processor to execute the actions specified in the software source code, they must be converted into processor instructions. There are many ways of conversion. Let us consider the most basic and common ones.

**Compilation** is the process of converting software source code into machine code (a set of processor instructions) that can be directly executed by the processor. During compilation, the compiler analyzes all the source code and creates an executable file that contains the machine code of the software. The executable file is machine-specific, thus a separate compilation process must be performed for each processor architecture. Programming languages that use compilation as a way to convert source code into processor instructions are called compiler-based languages.

One of the main advantages of compiler-based languages is their ability to generate time- and memory-optimized machine code. Most compiler-based programming languages require explicit declaration of variable data sizes and types. This allows to use computing system resources more efficiently and to detect typing errors at early stages of development, which helps to improve software reliability and security.

Software development in compiler-based languages can be more time-consuming because such languages usually require more rigorous syntax and language constructs. An error in the source code can lead to hard-to-detect compilation errors, which requires additional time to debug, find and fix the cause of the error. Compiler-based programming languages require the program code to be recompiled each time a change is made, which can be a time-consuming and costly process when dealing with large software. This results in reduced flexibility and speed of software development. Common compiler-based programming languages include C, C++, Rust, Go.



**Fig. 9.** Source code in the C programming language.

**Interpretation** is the process of executing source code by analyzing it and interpreting it sequentially by an interpreter during software execution. Unlike compiler-based programming languages, where the source code is pre-transformed into processor instructions and stored as an executable file, interpreted programming languages analyze and execute each syntactic unit of the source code directly at runtime. Programming languages that use interpretation as a way of translating source code into machine code are called interpreted languages.

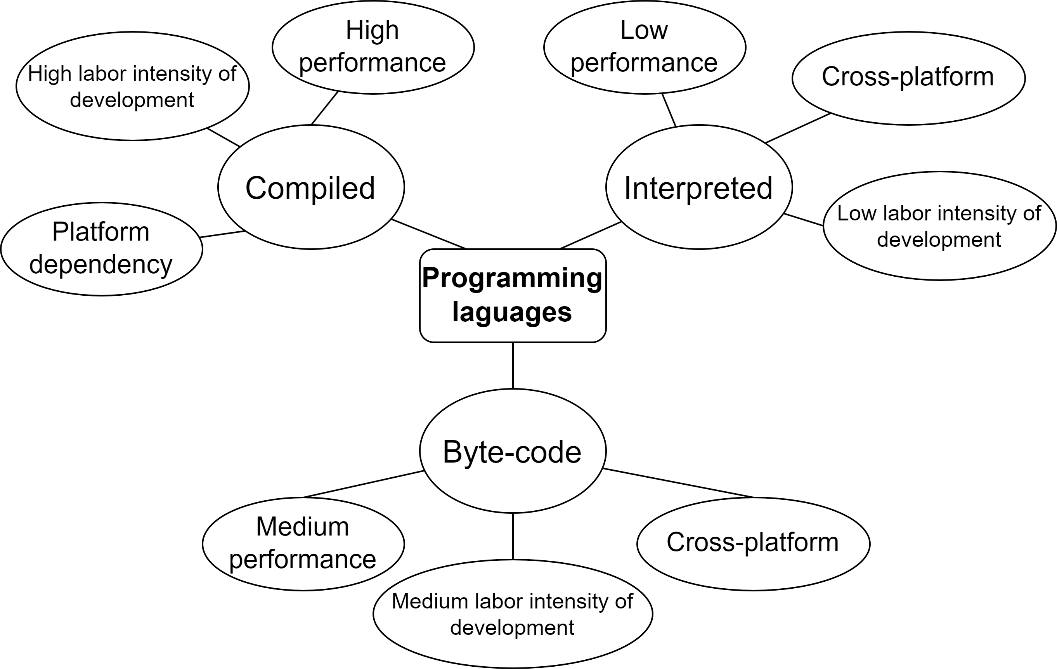
One of the main advantages of program interpretation is its platform independence. Interpreted programming languages in most cases can be executed on different platforms without the need to modify the source code. When interpreting software, it is easier to track and correct errors because the interpreter executes the code line by line. Interpreted programming languages usually support dynamic typing, which means that variable types can be defined at runtime.

However, software interpretation also has its disadvantages. One of the main disadvantages is low performance compared to compiler-based programming languages. Since the interpreter analyzes and executes the source code, the execution time can be longer than that of software compiled into machine code. This is especially noticeable when processing large amounts of data or performing highly loaded calculations. Common interpreted programming languages include Python, JavaScript, PHP, Ruby, SQL and Bash.

There are also **bytecode programming languages** that use both compilation and interpretation to convert source code into machine code. Bytecode programming languages represent the source code of a program in an intermediate form. The compiler converts the source code into a sequence of instructions that are low-level and intermediate for execution on a virtual machine or interpreter. This intermediate code is called bytecode because it is represented as a sequence of bytes.

One of the main advantages of using bytecode is platform independence. Building a virtual machine or interpreter that executes bytecode allows software to run on different platforms without changing the source code. Bytecode is also often used to improve software performance. It can be compiled in the most efficient way for a particular virtual machine or interpreter, resulting in more efficient software execution. Common bytecode programming languages include Java, C#, Kotlin and Scala.

Using the results of the analysis, a mind map of programming languages classification by the criterion of source code conversion into processor instructions was made, presented in Fig. 10.



**Fig. 10.** Mind map of programming languages classification.

According to the results of the analysis we can conclude that each of the presented types of programming languages has its own advantages and disadvantages. As a consequence, each of them is used in different areas of software development. The choice programming language type is determined by the task requirements.

* 1. Analysis of the Programming Languages under Research

Based on the conducted analysis of programming languages it is necessary to select those that potentially have the characteristics of speed and efficiency of development in their application. To solve the programming language selection problem, the evaluation criteria were compiled:

* performance - how fast the software developed with the use of programming language is executed on the computer system;
* relevance - how up-to-date the programming language is and what potential it has in the future for further prospective development of the software product;
* prevalence - the popularity of the language among the developer community and, as a consequence, the amount of available information and documentation on the Internet;
* number of integrated abstractions and syntax - a set of constructs, abstractions, paradigms and special tools that increase the efficiency of development using a programming language;
* ecosystem - availability of IDEs with automation of typical repetitive processes, availability of a compiler or interpreter without program errors, availability of an effective debugging system for the developed software product.

According to the compiled criteria the selection of modern and relevant programming languages was made. Below is an overview and analysis of the selected programming languages to evaluate their performance.

**Python** is an interpreted script programming language developed in the late 1980s by Guido van Rossum. It has a simple syntax and is the most popular programming language at the moment according to the TIOBE index [14]. One of the key advantages of Python is its ease of use. Python offers extensive built-in abstractions, which makes development in it highly efficient. Python is an interpreted language and as a consequence it is much slower compared to compiler-based languages, but there are various ways to optimize for speed. Python is known for its many libraries and frameworks that make it a very effective tool for a wide range of tasks. It has a well-developed ecosystem with a large number of development tools.

**Cython** is a programming language that is an add-on to Python with the ability to explicitly define data types. It allows developers to write high-performance code close to C that can be easily integrated with existing Python code. One of the main advantages of Cython is its performance. By explicitly defining data types and using static typing, Cython can synthesize CPU instructions that are optimized for performance. Cython is used extensively in projects that require a tradeoff between ease of development and performance. Cython is not as widespread as Python, but still has an audience and is growing in popularity. Cython provides the same integration and syntax as Python, adding the ability to use static typing.

**JavaScript** is a programming language developed by Netscape Communications Corporation in 1995. It is widely used to create interactive web pages and web applications and, along with HTML and CSS, is one of the primary languages for web development. JavaScript can also be used to develop server-side web applications using the Node.js platform. This allows to develop full-fledged software that can run on both client-side and server-side.

JavaScript is an interpreted language, but many built-in optimizations improve its performance. JavaScript is one of the most widely used programming languages and has a huge audience of developers. JavaScript provides many integrated abstractions and has an easy-to-understand syntax.

**C#** was developed by Microsoft in 2000 and is a key language for the ".NET" platform. C# combines the strengths of C++ with the convenience of Java, making it very popular among developers. One of the main advantages of C# is its object-oriented nature. It provides inheritance, encapsulation and polymorphism mechanisms to build modular and extensible programs. In addition, C# has built-in tools for automatic memory management, which makes development easier and eliminates many errors associated with memory misuse.

C# is slower relative to compiler-based languages, but modern solutions to optimize for performance make it quite efficient. C# is used to develop applications under Windows operating system and computer games on Unity platform. C# offers a wide set of integrated abstractions and has a syntax that is a combination of C++ and Java. C# is accompanied by an extensive ecosystem, including the integrated Visual Studio development environment and many libraries for various purposes.

**Kotlin** is a programming language developed by JetBrains in 2011 specifically for developing applications on the Java platform. Kotlin is a statically typed language that has a simple and clear syntax and is an extension of Java. One of the main advantages of Kotlin is its compatibility with Java. This allows developers to easily integrate Kotlin into existing Java projects and reuse existing source code. It has many improvements over Java. In addition, Kotlin provides extensive syntax features, making it easy to write safe and efficient code.

It is a bytecode language, compiles to Java bytecode and exhibits good performance but still lacks to the level of compiler-based languages. Kotlin is not as widely used as Java, but is getting popular rapidly.

**Rust** is a relatively new system programming language developed by Mozilla Research in 2010. First of all, Rust is known for its ability to improve security and avoid many common programming mistakes. One of the key advantages of Rust is its memory access control system, which prevents errors related to inconsistent memory accesses. This makes Rust especially safe for system software development, where low-level errors can lead to serious problems.

Rust also has an efficient type system and static analysis, which allows errors to be detected at compile time. This greatly simplifies debugging and improves the quality of the code being developed. In addition, Rust provides a mechanism for automatic memory management. Rust is a compiler-based programming language that shows excellent performance. Rust is still evolving and is not as widely used as other languages, but has potential applications in specialized areas. Has an evolving ecosystem with libraries and development tools.

**C** is a programming language developed in the early 1970s and is one of the most widely used languages in the software development industry. It is known for its capabilities and efficiency, as well as its broad support for various architectures and operating systems. One of the main advantages of C is its low level of abstraction from hardware, which allows developers to have full control over memory and the processor. This makes C particularly suitable for developing system software, embedded software, operating systems, and device drivers.

In this research, the C programming language is used as a benchmark programming language with the fastest performance and the widest distribution. Thus, it is not considered for the development of emulator of embedded systems due to the lack of support for object-oriented programming and a small number of integrated abstractions.

As a result of the analysis, seven programming languages were chosen, which will be used to implement the algorithms under research and comparative analysis of the results of their execution. It should be noted that some of the selected programming languages are used to implement emulators of the embedded systems discussed above. Thus, in further studies it will be possible to carry out a comparative analysis by the criterion of performance of the developed emulator of embedded systems on the programming language selected by the results of this research and existing solutions of emulators of embedded systems.

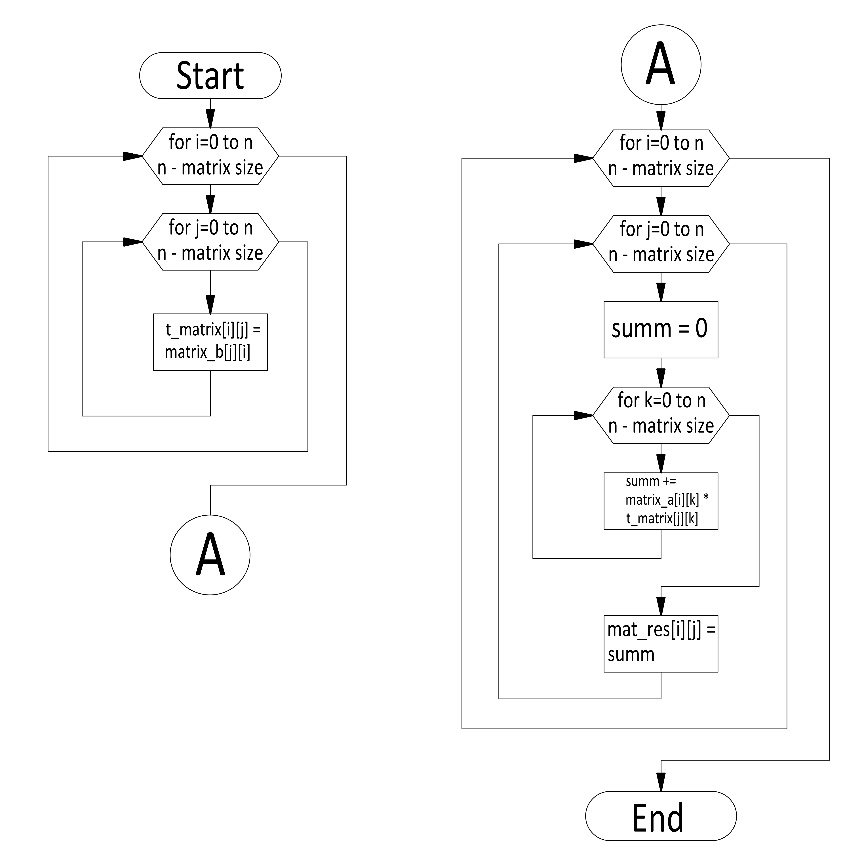
* 1. Analysis of the Researched Algorithms

Three algorithms were selected for the research of programming language performance. These operations are the most common operations performed by the processor. Hence, they will be performed most often in the emulator of embedded systems. It is necessary that the software developed in the programming language, in which the emulator will be implemented, performs these operations as fast as possible. The given algorithms do not consider auxiliary operations such as memory allocation, data reading, and output of results. Let us consider the selected algorithms in details.

**Matrix multiplication algorithm** is a process that consists of multiplying two matrices to obtain a third matrix, called the product matrix. The algorithm is implemented using loops, where each element of the product matrix is matched by the sum of the products of the elements of the corresponding rows and columns. The difference in the implementation is that the second matrix is transposed for more efficient memory usage. The flowchart of the matrix multiplication algorithm is shown in Fig. 11.

The matrix multiplication algorithm has a complexity of , where is the matrix side size, because it contains three nested loops. The time complexity of this algorithm depends on the size of the input matrices and can be significant for large matrices.

**Quicksort** is one of the most efficient methods of sorting data arrays. The quicksort algorithm is implemented by selecting a reference element from the original array and then dividing the array into two subgroups: elements smaller than the reference and elements larger than the reference. These subgroups are recursively subjected to a sorting process using quicksort until a final ordering is achieved. This process is called "partitioning".

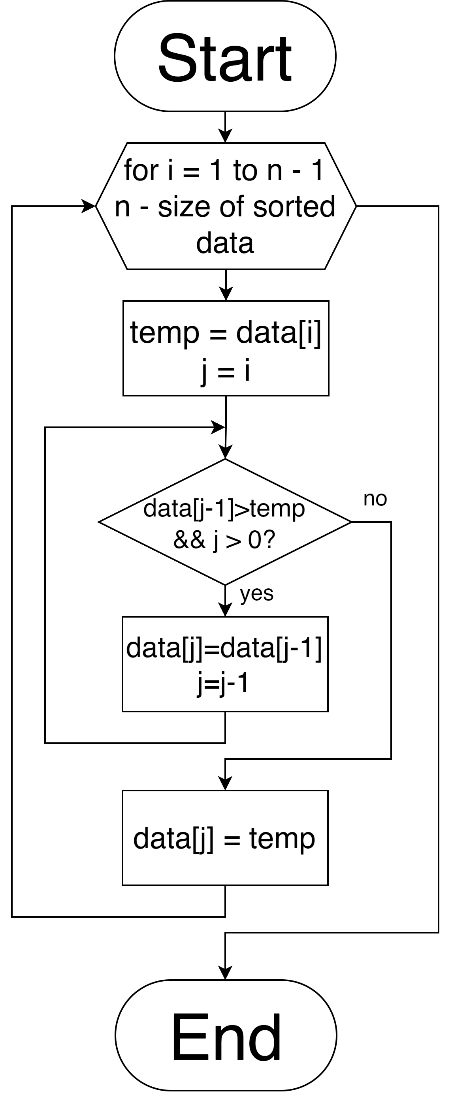


**Fig. 11.** Flowchart of the matrix multiplication algorithm.

The quicksort algorithm is highly efficient, having an average time complexity of , where is the number of elements in the array. However, in the worst case, its time complexity can reach , which happens when the reference element is unsuccessfully chosen or when the array is already ordered in ascending or descending order. The flowchart of the quicksort algorithm is shown in Fig. 12.

**The algorithm for finding prime numbers** solves the problem of finding all prime numbers up to a given integer . Prime numbers are natural numbers that are divisible by themselves and by 1. The rest of the natural numbers are called composite numbers. The only exception, 1, is not referred to either prime or composite. The solution of this problem is widely used in cryptography, data analysis and other applied problems.

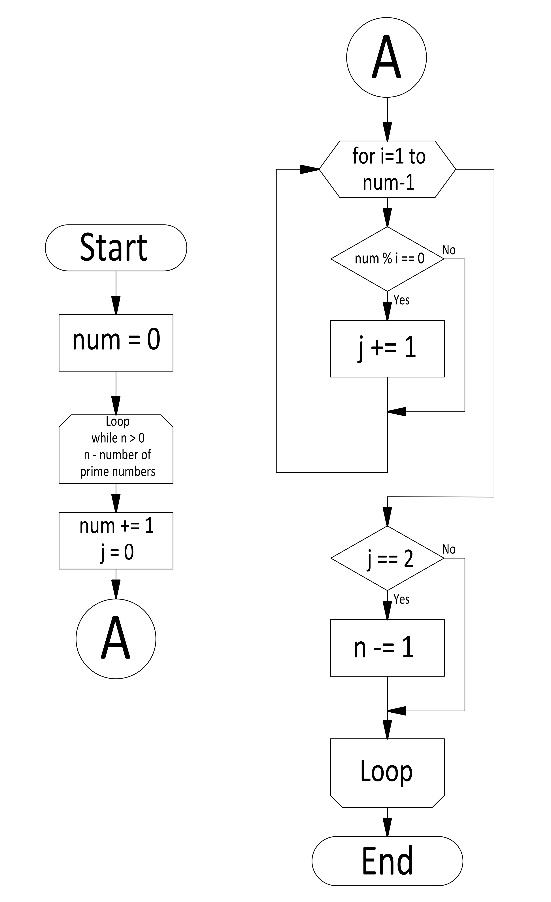
There are many algorithms optimized by the criterion of speed to find prime numbers: Sieve of Eratosthenes, Sieve of Sundaram, Sieve of Atkin, Wheel factorization, Base 2 Segmented factorization. For this study, the simplest algorithm for finding prime numbers was chosen - searching all numbers up to a given N and checking them for primality. This algorithm lacks obvious optimizations in terms of speed, such as discarding all even numbers after the number "2". The algorithm has a time complexity of , where is the number of prime numbers sought.



**Fig. 12.** Flowchart of the algorithm of quicksort.

The justification of this choice is to test the performance of software developed in a certain programming language. It is necessary to perform the task of loading the computer system to perform certain mathematical operations, but not the task of searching for prime numbers. For this reason, the developed algorithm does not save the found numbers except the last one. The flowchart of the developed prime number search algorithm is presented in Fig. 13.

As a result of the analysis, three algorithms were selected that contain the operations most frequently performed in the emulator of embedded systems. For each of the algorithms, the flowchart of the algorithm and its time complexity are presented. These algorithms were implemented in the programming languages discussed above and integrated into the "FPLB" software to test the speed of their execution.



**Fig. 13.** Flowchart of the brute force prime number search algorithm.

1. Description of the Research Method

The method for the research was designed to create a stable environment in which the effect of the programming language on the speed of execution of the algorithms could be evaluated. Standard scripts for testing programming languages for performance were used as a basis.

The software for testing was installed on a virtual machine with the distribution of the operating system of the Linux family Ubuntu 22.04. Parameters of the virtual machine:

* Processor AMD Ryzen 5 5600U with Radeon Graphics 2 cores;
* RAM 4 GB;
* ROM 60 GB;
* Video adapter is not used.

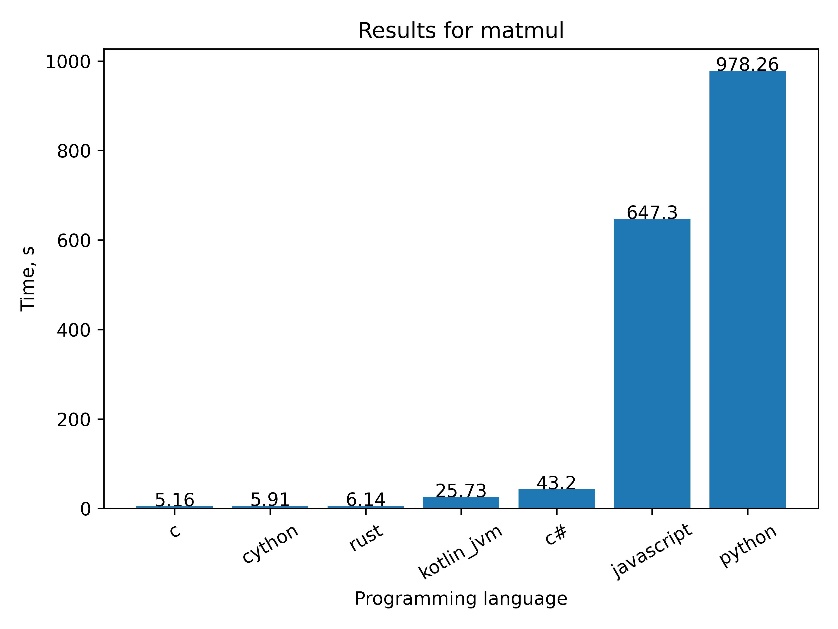
The testing is performed in a Docker container based on the Ubuntu Linux 22.04 family operating system. It allows the environment to be kept in the same stable state, which reduces the impact of various processes occurring on the computing system on which the testing takes place.

The "FPLB" testing software provides for the configuration of a sequence of tests in certain programming languages and certain algorithms. Thus, three configurations are created for each of the researched algorithms. The "FPLB" software also has the functionality of running a sequence of a specified number of the same tests. This allows to run many identical tests and calculate the arithmetic mean of the obtained results. The result is rounded to 2 decimal places. For each test a new set of input data of the same specified dimensionality is generated. Tests were carried out at 20 cycles of test repetitions. Based on statistical methods, this method of testing will give results close to real values.

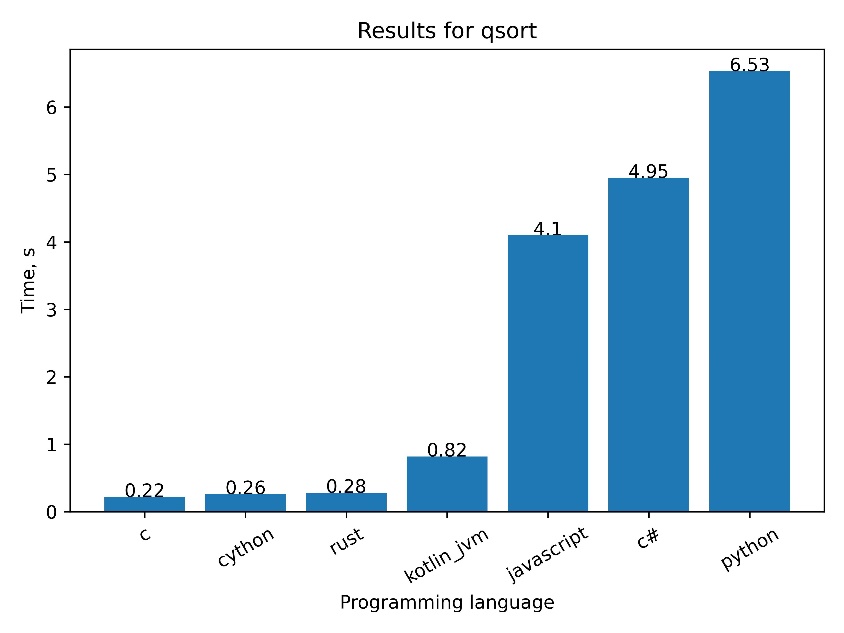
For the brute force prime number search algorithm, "FPLB" software generates the maximum amount of prime numbers to search in the range from 10000 to 15000. For the matrix multiplication algorithm, matrices of size 2000 by 2000 elements with values in the range from 0 to 65536 are generated. For the quicksort algorithm, data arrays of 1000000 elements with numbers in the range of 0 to 4294967294 are generated.

1. Analysis of the Research Results

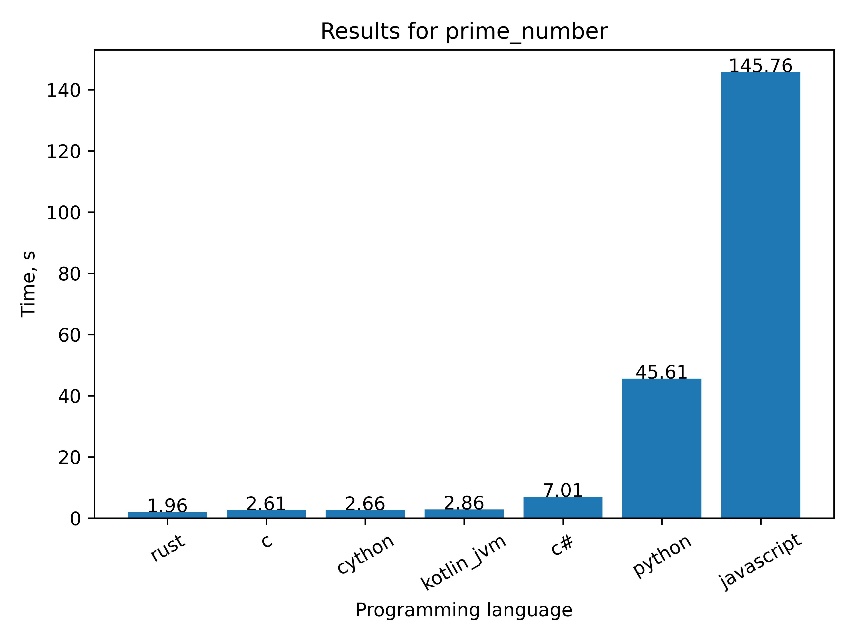
The result of this research is a set of bar graphs, with the vertical axis representing time in seconds and the horizontal axis representing programming languages. The resulting bar charts from the "FPLB" software for each of the studied algorithms are presented in Fig. 14 - 16.



**Fig. 14.** Matrix multiplication algorithm evaluation results.



**Fig. 15.** Quicksort algorithm evaluation results.



**Fig. 16.** Enumerative prime numbers search algorithm evaluation results.

The results of the research correlate with the results of the analysis of programming languages and their classification by the criterion of source code conversion into processor instructions. Compiler-based Rust and C languages show the best performance in the conducted research. Their results are approximately equal taking random error into account. The Cython programming language also shows high performance as it has a compilation mechanism similar to that used in the C programming language.

Bytecode programming languages Kotlin and C# show average speedup values. In the research of the algorithm for finding prime numbers, the Kotlin programming language managed to get a result comparable in speed to compiler-based programming languages. On the other hand, in the research of the algorithm of quicksort the C# programming language yielded to the interpreted programming language JavaScript with a small lag. Based on this, we can conclude that bytecode programming languages can show performance results in a wide range depending on the task at hand.

The interpreted programming languages Python and JavaScript showed the worst performance results in this research. The results of other programming languages are an order of magnitude higher than those of interpreted ones. It should be noted that in the research of the algorithm of searching for prime numbers by brute force Javascript showed the result almost 3 times slower than Python. At the same time, in other studies of algorithms JavaScript showed better results than Python. This result is presumably due to the use of the "BigInt" data type in the implemented algorithms, which allows storing numerical values larger than , as opposed to the standard Int data type. This is due to the fact that the investigated algorithms can obtain numerical values larger than those that fit into the standard Int data type. Presumably, the use of the BigInt data type requires large computational resources and, as a consequence, reduces the performance of software developed in the JavaScript programming language with the use of this data type. It is also worth noting that in this case the applicability of the JavaScript programming language for the development of emulator of embedded systems is a controversial issue, since modern processor architectures use machine words with the size of 64 bits, and there are registers with the size of 128 bits and more.

The analysis of the research results allowed to confirm the classification of programming languages according to the criterion of source code conversion into processor instructions. The peculiarities of the behavior of software developed in some programming languages were also revealed. It follows from the results of the research that the speed of programming languages can vary significantly depending on their peculiarities and optimizations. However, it should be noted that the speed of operation is not the only criterion when choosing a programming language, usability, availability of developers and other factors should also be taken into account.

It was revealed that the choice of programming language for software development should be carried out taking into account the specific requirements and tasks that the developed software product solves. However, it should be noted that the results of this research are relative and may depend on many factors, including the specific implementation of the programming language, code optimization, hardware and other conditions.

1. Discussion

The results of the research have contributed to the understanding of the impact of programming language choice on the execution speed of the software being developed for the emulator embedded systems development task. This section analyzes the results obtained, places them in the context of existing knowledge and current technological trends, and discusses possible practical applications of the study and future research directions.

An important aspect of the obtained results is the confirmation of the fact that software developed with the use of compiler-based programming languages has a performance characteristic that is orders of magnitude higher than that of software developed with the use of interpreted languages. Thus, interpreted programming languages cannot be applied to implement a modern emulator of embedded systems that meets the criteria of speed and efficiency of its support as a software product.

It is worth noting that the Cython programming language showed impressive results. Despite the fact that in the process of execution of some parts of the software there are calls to the Python virtual machine, which reduces its performance, the results at the level of compiler-based C programming language Rust were demonstrated. Thus, the Cython programming language is a promising language for the development of emulator of embedded systems due to its high performance, which is a consequence of the compilation of the source code, simplicity, prevalence, the number of integrated abstractions and syntax, as well as the presence of an extensive ecosystem that is present around the Python language.

It is also worth noting that JIT compilation was not considered or utilized in the conducted research. It is an effective tool to entrain the performance of interpreted and bytecode programming languages. This is due to the complex behavioral algorithm of JIT-compilation, which requires in-depth research and analysis. In this way, it will be possible to explicitly make an evaluation of the execution speed of software in different programming languages considering JIT-compilation. Also, object-oriented programming techniques have not been addressed in the research. This is due to the different implementation of object-oriented programming in the internal architecture of different programming languages. The impact of applying object-oriented programming techniques on the performance of software in different programming languages requires a separate research.

The results of this research also correlate with the results obtained in other works [15, 16, 24]. Software execution performance is crucial to the success of computer systems and applications in the modern world. With the increasing number of workloads on modern computing systems, it is important to execute tasks successfully and timely. Thus, the importance of the speed of programming languages in the modern world can hardly be overemphasized, as the further development of information technology directly depends on it.

It is noteworthy that some programming languages showed slightly different results executing different algorithms. This opens prospects for new studies using more algorithms, which will reveal more accurate patterns in understanding performance of software developed in the studied programming languages. In general, the conducted research opens new prospects for the development of modern emulators of embedded systems and the development of new methods to improve their performance.

1. Conclusion

In this paper a research aimed at analyzing the performance of programming languages in the context of the task of emulation of embedded systems software has been performed. The problem of selecting a programming language for software product development has been revealed. Thus, the choice of programming language is a complicated and multifaceted task that requires a complex approach and the use of various methods of analysis.

The comparative analysis and research of existing solutions of emulators of embedded systems was carried out. The need to develop an emulator that will have high performance characteristics and ease of implementation of new modules has been revealed. The need for comparative testing of the performance of different programming languages has been revealed. To test the performance of programming languages, a software "FPLB" in the Python programming language has been developed. The algorithm of the developed software is described in the paper in details.

The analysis of the software execution on the computer system has been carried out. The classification of programming languages by the criterion of conversion of the source code into processor instructions has been made. To choose programming languages, for which the research was carried out, the criteria for evaluation of programming languages have been made. As a result of the analysis, seven programming languages have been chosen for the research, for which the implementation of the algorithms under research was carried out and the comparative analysis of the results of their execution has been made.

To conduct the research, we selected and analyzed the algorithms that were implemented using the selected programming languages. As a result of this analysis, three algorithms were selected that contain operations that are most frequently performed in the emulator of embedded systems. For each of the algorithms, a block diagram of the algorithm and its time complexity is presented. The method of conducting the research using the developed software "FPLB" is developed and described.

The results of the research were analyzed. The analysis of the research results allowed to confirm the classification of programming languages according to the criterion of conversion of source code into processor instructions. Based on the results of this research it was revealed that the choice of programming language for software development should be carried out taking into account specific requirements and tasks that the software product being developed solves.

The results of this research can be useful for engineers and developers working in the field of embedded systems design and software development for them. The obtained recommendations will help to increase the efficiency of software development and testing for embedded systems, as well as to reduce possible risks and problems associated with incorrect emulator behavior.

In the future, based on the results of this work, it is possible to conduct additional research in the field of emulator optimization for embedded systems by the criterion of performance. In general, the presented work is an important contribution to the field of emulator development for embedded systems and can become a useful reference material for specialists engaged in this problem.

Acknowledgements

The results were partially obtained within the project «PRIOR/SN/NU/22/SP2/5» under the Development Program of Bauman Moscow State Technical University as part of the Priority-2030 federal academic leadership program.

References

1. Artemiev, B.V., Vlasov, A.I., Juravleva, L.V.: Quality estimation method in advanced software systems. AIP Conference Proceedings 2467(050011), (2022).
2. Zobov, O.V., Vlasov, A.I., Juravleva, L.V.: The building data collection devices problems analysis in the industrial IoT paradigm. AIP Conf. Proc. 2402(050041), (2021). DOI: 10.1063/5.0072134.
3. Efimov, S.N., Terskov, V.A., Galushin, P.V., Yarkov, K.V.: Reliability model of multiprocessor hardware-software complexes of real-time control systems with multi-version software. Herald of the Bauman Moscow State Technical University. Series Instrument Engineering 4 (137) (2021). DOI: 10.18698/0236-3933-2021-4-41-58.
4. Zaborovskiy, N.V., Tormasov, A.G.: Modeling of multithreaded program execution and the method of static code analysis for race conditions. Journal of Applied Informatics 4(34), 105-110 (2011).
5. Peter Boonstoppel, CristianCadar, Dawson Engler, RWset: Attacking Path Explosion in Constraint-Based Test Generation. Lecture Notes in Computer Science book series 4963, 351–366 (2008).
6. Vlasov, A.I., Selivanov, K.V., Kovalev, I.O.: Investigation of Quality and Identity of Similar Electronic Components from Different Manufacturers. International Conference on Industrial Engineering, Applications and Manufacturing, ICIEAM 2023, pp. 333-338. IEEE, Sochi, Russian Federation (2023). DOI: 10.1109/ICIEAM57311.2023.10139051.
7. Sepulveda Rodriguez, L. E., Chavarro-Porras, J.C., Sanabria-Ordonez, J.A., et al.: A Survey of Virtualization Technologies: Towards a New Taxonomic Proposal. Ingenieria E Investigacion 42(3), e97363 (2022).
8. Selvesyuk, N.I., Ostrovskiy, A.S., Rusanov, P.V., Komakhin, M.O.: Object-oriented design of the system of emulation of program interface of the application in the tasks of ensuring information security. Information Science and Control Systems 2(56), (2018). DOI: 10.22250/isu.2018.56.3-13.
9. Zhang, B.,et al.: A Survey on I/O Virtualization and Optimization. Fifth Annual China Grid Conference 2010, pp. 117-123, IEEE, Guangzhou, China (2010). DOI: 10.1109/ChinaGrid.2010.54.
10. Dileep, K.P., Devesh, G., Raghavendra, R., Suman, A., Srikanth, S.: Verification of Linux device drivers using device virtualization. 2nd International Conference on Computing for Sustainable Global Development, INDIACom2015, pp. 694-698, IEEE, New Delhi, India (2015).
11. Gladkikh, A.A., Vlasov, A.I., Uzenkov, D.A., Fatkhutdinov, T.M.: Dynamic modification of embedded devices internal firmware for solve reverse engineering problems. All-Russian Scientific and Technical Conference "Problems of development of advanced micro- and nanoelectronic systems" 2, 63-69 (2021). 10.31114/2078-7707-2021-2-63-69.
12. Gill, S.: The Diagnosis of Mistakes in Programmes on the EDSAC. Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences 206(1087), 538-554 (1951).
13. Jakob Engblom, Guillaume Girard, Bengt Werner: Testing Embedded Software using Simulated Hardware, Conference ERTS’06, Jan 2006, Toulouse, France. ffhal-02270472f (2006).
14. Index TIOBE, https://www.tiobe.com/tiobe-index/, last accessed 2024/03/03.
15. Stefan Marr, Benoit Daloze, Hanspeter Mössenböck: Cross-language compiler benchmarking: are we fast yet? In Proceedings of the 12th Symposium on Dynamic Languages, DLS 2016, pp. 120–131, Association for Computing Machinery, New York, NY, USA (2016).
16. Ivanov, S., Ilin, D., Bolshakov, I.: Benchmark of Programming Languages. Bulletin of Chuvash University 3, 222–227 (2017).
17. Qemu emulator, https://www.qemu.org/, last accessed 2024/03/03.
18. Kopycat emulator, https://kopycat.ru/, last accessed 2024/03/03.
19. Aristov, R.S., Gladkikh, A.A., Davydov, V.N., Komakhin, M.O.: Developing Software Kopycat Platform for Emulation of Complex Computing Systems. Nanoindustry S(89), 350-352 (2019). DOI 10.22184/NanoRus.2019.12.89.350.352.
20. Unicorn emulator, https://www.unicorn-engine.org/, last accessed 2024/03/03.
21. CAD Proteus, https://www.labcenter.com/, last accessed 2024/03/03.
22. Integrated development environment Keil, URL: https://www.keil.com/, last accessed 2024/03/03.
23. Software FPLB, https://github.com/eleven5150/mips-emulation/tree/main/benchmarking, last accessed 2024/03/03.
24. Korovin, I.V., Pulkin, I.A., Veranyan, A.S.: Research of Speeds of Performance of Basic Mathematical Problems of Popular Programming Languages. Economics and quality of communication systems 3(13), 63-68 (2019).
25. Frolov, V.A., Galaktionov, V.A., Sanzharov, V.V.: Investigation of the RISC-V. Trudy ISP RAN/Proc. ISP RAS32(2), 81-98 (2020). DOI: 10.15514/ISPRAS-2020-32(2)-7.