Гладких А.А., Кутаев К.С.

**Применение современных языков программирования для решения задачи разработки эмулятора встраиваемых систем**

**Аннотация:** Работа посвящена исследованию применения современных языков программирования для решения задачи разработки эмулятора аппаратных платформ, применяемого для отладки разрабатываемой программной части встраиваемых систем. Авторы проводят исследование характеристики быстродействия современных языков программирования в выполнении определенных операций и алгоритмов, наиболее часто используемых в процессе эмуляции встраиваемых систем с помощью разработанного для проведения исследования программного средства. В статье также обсуждаются характеристики существующих эмуляторов выстраиваемых систем и предлагаются рекомендации по разработке эмулятора, обладающего более высокими характеристиками быстродействия и эффективности поддержки его как программного продукта. Полученные результаты могут быть полезными для инженеров и разработчиков, работающих в области проектирования встраиваемых систем и разработки программного обеспечения под них.

**Ключевые слова:** встраиваемая система, эмулятор, язык программирования, процессор, микроконтроллер.

Gladkikh A.A., Kutaev K.S.

**Abstract:** The work is devoted to the study of the use of modern programming languages to solve the problem of developing a hardware platform emulator used for debugging the developed software of embedded systems. The authors conduct a study 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 article also 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 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.

**Keywords:** embedded system, emulator, programming language, processor, microcontroller.

**Введение**

В современном мире встраиваемые системы являются неотъемлемой частью большинства отраслей жизнедеятельности человека [1]. Встраиваемая вычислительная система представляет собой любую вычислительную программно-аппаратную систему, не являющуюся системой общего назначения, такой как персональный компьютер или сервер. Их применение охватывает самые разнообразные сферы, включая автомобильную промышленность, медицину, энергетику, системы безопасности, промышленное производство, бытовую технику и т.д. Технологии Интернета вещей (IoT) также в значительной мере опираются на встраиваемые системы для создания сети устройств, которые можно удаленно контролировать и управлять [2].

В процессе разработки программного обеспечения (ПО) для встраиваемых систем часто возникает потребность в его отладке и тестировании. Выполнять эти операции на разрабатываемой встраиваемой системе в большинстве случаев невозможно в связи с отсутствием физического доступа к ней или недостатка на ней необходимых аппаратных частей, которые ещё находятся в процессе проектирования или производства. Даже при полностью спроектированной аппаратной части встраиваемой системы не всегда есть возможность получать подробную информацию о всех процессах, происходящих в ней, которые необходимо отлаживать и тестировать. Это связано с тем, что в современных встраиваемых системах используются множество аппаратных компонентов, между которыми происходит различная коммуникация, и отслеживать состояния одновременно во всех компонентах затруднительно [3]. Данные процессы коммуникации также могут зависеть от внешних событий, и тогда сложность отладки значительно возрастает [4].

Запуск ПО встраиваемых систем на вычислительной системе разработчика (персональном компьютере) также может быть затруднительным. Одной из причин является различие процессорных архитектур вычислительной системы разработчика и целевой системы, под которую ведется разработка. Встраиваемые системы зачастую используют специализированные процессоры или микроконтроллеры, которые ориентированы на выполнение узкоспециализированных задач с низким энергопотреблением [1]. Одним из способов решения данной проблемы является использование эмулятора - программного средства, которое имитирует (эмулирует) работу вычислительной системы и ее окружения в другой вычислительной системе, чтобы имитированное поведение максимально точно соответствовало поведению настоящей вычислительной системе.

**Работа посвящена** исследованию применения современных языков программирования для решения задачи разработки эмулятора аппаратных платформ, применяемого для отладки разрабатываемой программной части встраиваемых систем в условиях, когда отладка на физической части системы затруднительна или невозможна. В последние десятилетия встраиваемые вычислительные системы получили широкое распространение, поскольку все больше процессов как в повседневной жизни, так и в промышленности требуют высокой степени автоматизации и интеграции [2]. Таким образом, эффективность разработки ПО для встраиваемых систем становится критическим аспектом для их производства.

**Объектом исследования** выступают современные средства и методы разработки ПО, в частности популярные на момент написания данной статьи императивные языки программирования. Императивный язык программирования — это формальный язык, который используется для написания компьютерных программ, определяющих последовательность команд, выполняемых процессором. Основная идея императивного программирования заключается в том, чтобы указывать вычислительной системе, какие операции нужно выполнить и в каком порядке.

**Предметом исследования** является быстродействие современных императивных языков программирования в контексте задачи эмуляции ПО встраиваемых систем, а также их возможности для эффективной поддержки и разработки новых составных частей эмулятора встраиваемых систем.

**Целью работы** является исследование характеристики быстродействия современных языков программирования в выполнении определенных операций и алгоритмов, наиболее часто используемых в процессе эмуляции встраиваемых систем. Результатом является программное средство для проведения исследования языков программирования для определения быстродействия выполнения заданных алгоритмов, а также предложения по использованию средств разработки ПО для реализации современного эмулятора встраиваемых систем, отвечающего критериям быстродействия и эффективности поддержки его как программного продукта.

**Актуальность исследования** обусловлена развитием информационных технологий и увеличения интеграции встраиваемых систем во многие сферы промышленности и повседневной жизни. Встраиваемое ПО становится все более сложным и получает все более широкую функциональность. В связи с этим растет и количество процессов, происходящих в ПО встраиваемых систем. Это приводит к экспоненциальному увеличению возможных путей исполнения ПО [5]. Проводить тестирование, отладку и анализ исполнения такого ПО становится сложнее. С увеличением интеграции встраиваемых систем в широкий спектр процессов автоматизации, эффективная разработка и тестирование ПО становятся критическими составляющими разработки надежных, отказоустойчивых и безопасных встраиваемых систем.

Также важно отметить, что в промышленности, где встраиваемые системы имеют широкое применение, небольшие отклонения в ожидаемом поведении устройств могут привести к серьезным последствиям, включая повреждение инфраструктуры и нанесения вреда человеку.

Одним из возможных способов повышения эффективности проведения тестирования и отладки является запуск ПО в эмулирующей среде. Эмуляторы обладают широким спектром применения [6]. Они используются, в частности, для проведения динамического анализа выполнения ПО в рамках проведения сертификационных испытаний [7], для тестирования производительности [8], для подтверждения корректности поведения устройств или программ [9], а также для решения задач обратной разработки [10].

Использование эмуляции как метода тестирования и отладки ПО существует уже довольно давно, начиная с появления первых вычислительных систем [11]. Оно имеет ряд преимуществ по сравнению с использованием физического оборудования. При запуске эмуляции ПО для тестирования загружается путем его записи непосредственно в эмулируемую память. Таким образом, нет необходимости загружать ПО по последовательной линии или сети и запускать мониторинг состояния целевой встраиваемой системы. Эмулируемая встраиваемая система управляется по тому же интерфейсу передачи команд, что и реальная физическая система. Следовательно, нет необходимости физического доступа к целевой встраиваемой системе.

За счет того, что эмулятор запускается на автоматизированном рабочем месте (АРМ) разработчика, есть возможность создавать сценарии тестирования и автоматизировать их запуск. Поскольку тестирование выполняется путем запуска обычного процесса (эмулятора) на операционной системе (ОС) АРМ, запуск наборов тестов (множества отдельных тестовых случаев) можно произвести одновременно на нескольких АРМ. Детерминизм эмуляции упрощает проведение регрессионного тестирования – это проверка ранее протестированной части ПО, позволяющая убедиться, что внесенные изменения не повлекли за собой появления дефектов в той части ПО, которая не менялась. Любое изменение результатов выполнения по сравнению с предыдущими выполнениями других версий того же ПО может быть вызвано только различиями в тестируемом ПО, а не аппаратными сбоями или другими случайными изменениями [12].

В целом, применение эмулятора встраиваемых систем с характеристиками быстродействия и эффективности поддержки его как программного продукта представляет собой важный шаг в повышении качества разрабатываемых систем как промышленного, так и повседневного применения.

**Обзор литературы**

​Разработка эмулятора встраиваемых систем является нестандартной задачей в связи с его малой распространенностью и малым количеством существующих решений. При проектировании такого ПО часто встает вопрос о том какой язык программирования использовать. Для разработки современного ПО применяется множество языков программирования. Согласно рейтингу на индексе TIOBE в настоящее время около 50-ти языков программирования занимают в сумме 70% доли рынка [13]. Очевидно, что среди них нет универсального языка. У каждого из них есть своя область применения, своя философия, особенности, различные диалекты, фреймворки и множество других характеристик, по которым можно проводить их сравнение.

Исследования показывают, что правильный выбор средств и инструментов при разработке ПО оказывает существенное влияние на процесс разработки, качество результата и возможность дальнейшей поддержки продукта, особенно для различных критических систем [14]. Хотя большинство языков являются и универсальными, но некоторые из них являются более подходящими для конкретных целей, нежели другие [15]. Для уменьшения числа вариантов при выборе языка программирования необходимо отталкиваться от целей, которые достигаются за счет реализации данного ПО. Для этого необходимо оценить набор критериев, которыми должен обладать разрабатываемый продукт.

Так как язык программирования – это абстрактное описание вычислительного процесса, то объективно измерить можно только сам процесс и его результаты. Суть подобных методов сравнения языков программирования заключается в том, что оценивается код программы, написанной на разных языках. При этом следует учитывать разницу между языком программирования и программой при использовании этого метода и интерпретации результатов [15].

При проведении сравнения языков программирования следует использовать только базовые синтаксические конструкции языка, для достижения максимально объективной оценки производительности. Это связано с наличием в рассматриваемых языках уникальных синтаксических абстракции, принцип работы которых во внутреннем устройстве языка неочевиден или неизвестен, и таким образом не может быть точно воспроизведен в других языках программирования.

Таким образом, выбор языка программирования для написания современного ПО представляет собой сложную и многогранную задачу, требующую комплексного подхода и использования различных методов анализа. Однако, разработка методик и программных средств для сравнения быстродействия языков программирования является важным направлением для проектирования современного ПО, отвечающего критериям быстродействия и эффективности поддержки его как программного продукта.

При проектировании современного эмулятора встраиваемых систем необходимо выявить критерии, которым он должен соответствовать. Для этого проведем обзор и анализ существующих решений и аналогов эмулятора встраиваемых систем. ­­­­­

**Qemu (Quick Emulator)** – инструмент с открытым исходным кодом, который используется для эмуляции различных архитектур. Qemu имеет большое сообщество разработчиков и, как следствие, хорошую поддержку. Поддерживает два режима эмуляции: пользовательский режим, в котором происходит полная эмуляция системы, включая процессор и периферию, и системный режим, в котором происходит только трансляция инструкций и системных вызовов эмулируемой системы. На рисунке 1 приведен пример работы Qemu в системном режиме, где происходит эмуляция операционной системы ReactOS.

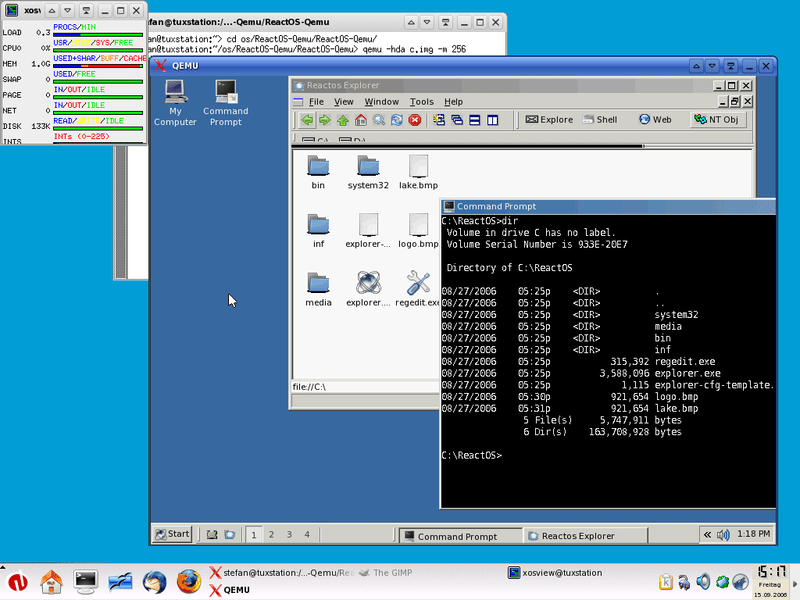


Рисунок 1 - Эмуляция операционной системы ReactOS в эмуляторе Qemu

Qemu обладает такими положительными характеристиками, как наличие интерфейса RSP GDB, моделирование на основе логического соединения шин и высокое быстродействие, так как написан на языке С. С другой стороны, это создает большой недостаток в виде высокой сложности написания новых модулей (архитектур, периферии и т.д.).

**KOPYCAT** – отечественный эмулятор аппаратных платформ, позволяет осуществлять низкоуровневую программную эмуляцию произвольных аппаратных систем и их отладку через интерфейс GDB, поддерживаемый большинством средств разработки. Реализован на языке программирования Kotlin (Java), главным преимуществом которого является эффективность разработки новых решений. На рисунке 2 представлена архитектура эмулятора для эмуляции микроконтроллера (МК) серии STM32.

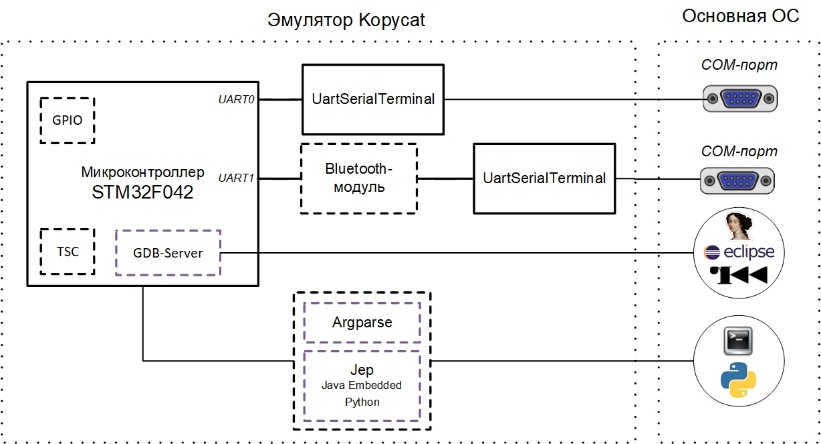


Рисунок 2 – Архитектура эмулятора KOPYCAT для эмуляции МК серии STM32

Одним из конкурентных преимуществ KOPYCAT является наличие удобного комплекта для разработки ПО и документации для создания новых вычислительных ядер и периферийных устройств. При этом возможно два подхода для создания новых модулей: первый — это структурный подход, второй — поведенческое описание. При структурном описании в файле в формате JSON указываются установленные модули и соединения между ними. Также структурное описание допустимо проводить и с помощью языка программирования Kotlin/Java. При поведенческом же описании в модуль могут быть добавлены обработчики записи или чтения с определенных адресов (к которому будет подключен модуль). Внутри обработчика при этом задается, каким образом модуль должен отрабатывать данное событие. Поведенческое описание может быть задано только на языке программирования Kotlin/Java.

**Unicorn** – это легковесный, мультиплатформенный и мультиархитектурный эмулятор процессора. Данный эмулятор отличается в своей архитектуре от остальных представленных в списке наличием только базовой функциональности. Он не эмулирует работу всей программы или целой ОС. Он не поддерживает системные команды (такие как открытие файла, вывод символа в консоль и т. д.). Разработчику необходимо самостоятельно проводить разметку памяти эмулируемого ПО и загружать данные в нее. На рисунке 3 представлен графический интерфейс эмуляции BIOS в Unicorn.

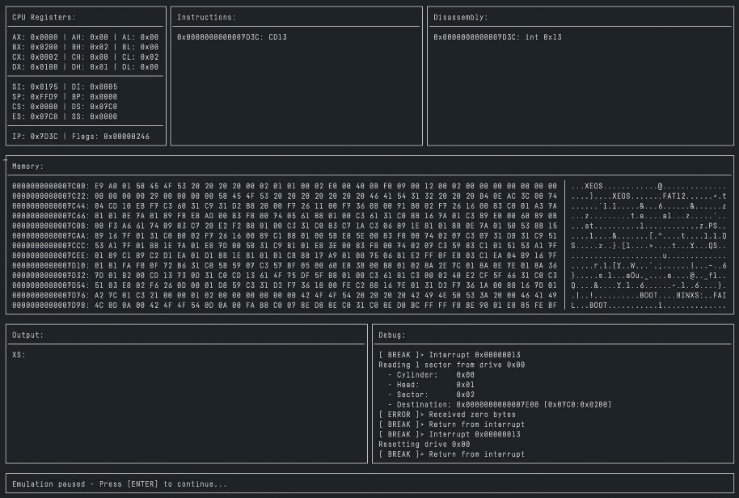


Рисунок 3 – Эмуляция BIOS в Unicorn

К недостаткам данного эмулятора относится отсутствие встроенных абстракций и инструментов по работе ПО, настройка и использование могут быть сложными для неопытных пользователей, отсутствие системных команд, сложность реализации новых модулей эмулятора. Таким образом для работы с Unicorn требуется высокая квалификация специалиста.

**Proteus** – пакет программ для автоматизированного проектирования электронных схем, отличительной чертой которого является возможность моделирования работы программируемых устройств. Эмуляция МК семейства PIC16 представлена на рисунке 4. Эмуляция в Proteus, в отличие от Qemu происходит на уровне электрических сигналов, что значительно замедляет ее скорость.

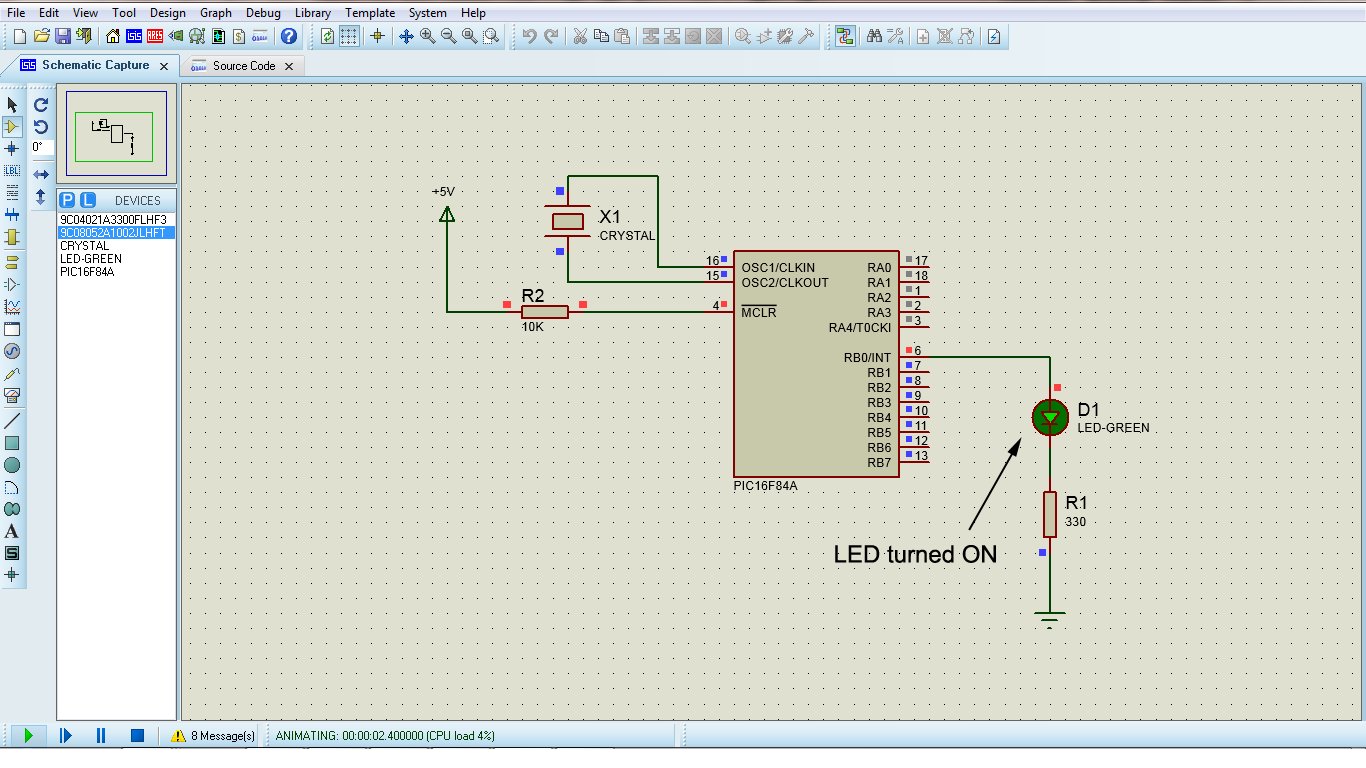


Рисунок 4 - Эмуляция работы микроконтроллера семейства PIC16 в Proteus

Является проприетарным ПО, в отличие от описанных выше эмуляторов, но присутствует возможность написания собственных модулей на C++. Эффективность их разработки мала в связи с отсутствием подробной документации, слабой поддержкой в сообщества разработчиков и спецификой языка С++.

**Эмуляторы, встроенные в IDE** – некоторые IDE, например ARM Keil имеют встроенный эмулятор аппаратных платформ (рисунок 5). Такие эмуляторы имеют слабую поддержку сообщества разработчиков, низкое быстродействие и практически неприменимы в задачах разработки встраиваемых систем.

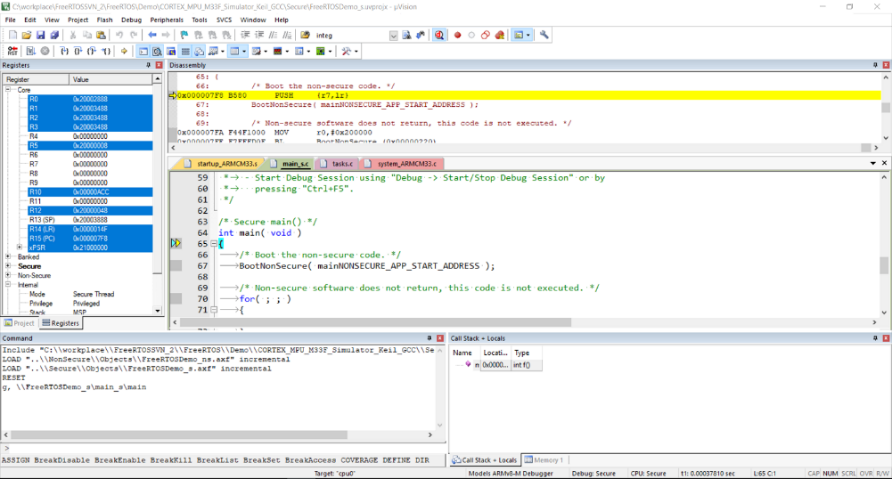


Рисунок 5 - Эмуляция работы ядра ARM Cortex-M33 в IDE Keil uVision

Это связано с отсутствием в них возможности эмулировать периферийные устройства, которые являются обязательной составляющей любой встраиваемой системы.

Сравнение существующих решений и аналогов эмулятора встраиваемых систем представлено в таблице 1. Значения качественных характеристик приведены относительно элементов этой таблицы.

Таблица 1. Сравнение инструментов эмуляции встраиваемых систем и моделирования электрических схем

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Программное средство** | **Быстродействие** | **Сложность реализации**  **новых модулей** | **Интерфейс GDB** | **Моделирование электрических сигналов** |
| Qemu | Высокое | Высокая | Есть | Нет |
| KOPYCAT | Среднее | Низкая | Есть | Нет |
| Unicorn | Высокое | Высокая | Есть | Нет |
| Proteus | Низкое | Высокая | Нет | Есть |
| Keil IDE | Низкое | Отсутствует | Есть | Нет |

В итоге рассмотрения данного вопроса можно сделать вывод, что существующие решения для проведения эмуляции встраиваемых систем не обладают характеристикой быстродействия и возможностью простой реализации новых модулей одновременно. При высокой скорости работы таких языков как C и C++ они обладают малым количеством встроенных средств и абстракций, слабо развиты и часто имеют не интуитивное поведение. Таким образом возникает потребность в разработке эмулятора, который будет обладать высокими характеристиками быстродействия и удобством реализации новых модулей.

**Материалы и методы**

Ключевым методом исследования является проведение тестирования быстродействия выполнения определенных алгоритмов различными языками программирования, сравнение результатов проведенных экспериментов.

Для проведения тестирования быстродействия языков программирования было разработано программное средство с необходимым для тестирования окружением, в проектировании которого были использованы исследования, представленные в работе «Cross-Language Compiler Benchmarking» [14]. Программное средство для тестирования позволяет в автоматизированном режиме производить запуск различных тестов для заданных языков программирования. Для его разработки был выбран язык программирования Python. Он имеет большую поддержку и активное сообщество разработчиков, что обеспечивает наличие множества ресурсов, документации, библиотек и инструментов для разработки. В качестве окружения для исполнения алгоритмов была выбрана контейнерная платформа Docker. Это распространенный легкий и автономный выполняемый контейнер, который включает в себя всё необходимое окружения для запуска ПО, в том числе библиотеки, системные инструменты, код и среду выполнения. Он позволяет поддерживать окружение в одном и том же стабильном состоянии, что позволяет снизить влияние различных процессов, происходящих на вычислительной системе, на которой происходит тестирование.

Для оценки быстродействия выполнения тестируемого ПО была использована методология, предложенная в исследовании «Сравнительное тестирование языков программирования» [15]. Значение времени исполнения тестируемого ПО на определенном языке программирования измеряется с помощью ПО «time». Это стандартная unix-утилита, которая выводит на вывод статистическое сообщение об использованном времени запущенным ПО.

Выбор алгоритмов, скорость выполнения которых будет тестироваться, был основан на работе «Исследование скоростей выполнения базовых математических задач популярных языков программирования» [16]. Так как целью работы является исследование характеристики быстродействия языков программирования в выполнении алгоритмов, наиболее часто используемых в процессе эмуляции встраиваемых систем, то необходимо выбрать те алгоритмы, которые содержат в себе операции, чаще всего используемые в работе эмулятора. Этими операциями являются чтение и запись данных в память, а также базовые арифметические операции. Это связано со спецификой работы современных процессорных архитектур. Таким образом требуются алгоритмы, содержащие в себе подобные операции.

Визуализация результатов была осуществлена с использованием метода, применяемого в исследовании «Cross-Language Compiler Benchmarking» [14]. Для этого была использована библиотека для языка Python «MatPlotLib». Она позволяет проводить визуализацию данных двумерной и трёхмерной графикой. Таким образом можно наглядно оценить соотношение скоростей выполнения алгоритмов различными языками программирования.

**1. Основы исполнения программного обеспечения**

Компьютерная программа с точки зрения процессора – это последовательность инструкций, описывающие действия, которые необходимо совершить над набором данных. Данные инструкции могут быть записаны в виде мнемонических кодов на языке Ассемблера. Все языки Ассемблера являются машинно-зависимыми, то есть ориентированы на определенный тип процессорной архитектуры. Пример мнемоник на языке Ассемблера архитектуры ARM представлены на рисунке 6.

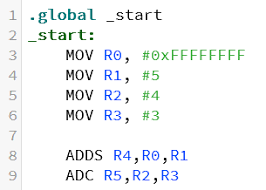


Рисунок 6 – Мнемоники инструкций на языке Ассемблера архитектуры ARM

С точки зрения разработчика программа – это синтаксическая единица, которая соответствует правилам определённого языка программирования, состоящая из определений и операторов или инструкций, необходимых для определённой функции, задачи или решения проблемы. Представлена она в виде текстового файла с набором исходного кода. Пример исходного кода на языке C представлен на рисунке 7.

В конечном итоге исходные код программы преобразуется в процессорные инструкции. Способов преобразования существует множество. Рассмотрим самые основные и распространенные из них.

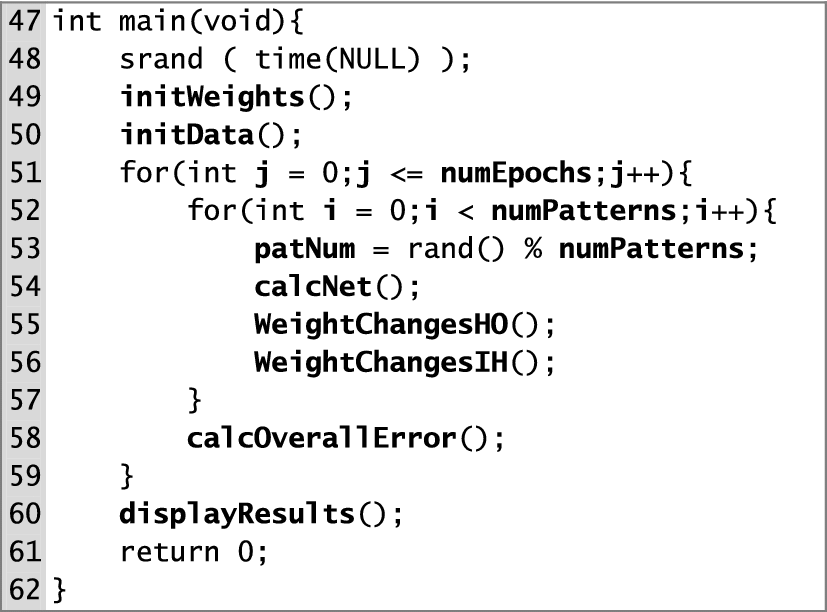


Рисунок 7 – Исходный код на языке программирования C

Компиляция – это процесс преобразования исходного кода программы в машинный код, который может быть непосредственно выполнен процессором. Во время компиляции, компилятор анализирует весь исходный код и создает исполняемый файл, который содержит машинный код программы. Исполняемый файл является машинно-зависимым, таким образом для каждой архитектуры необходимо проводить отдельный процесс компиляции. Языки программирования, которые используют компиляцию как способ трансляции исходного кода в процессорные инструкции называют компилируемыми языками.

Одним из главных преимуществ компилируемых языков является их способность генерировать оптимизированный по времени и памяти машинный код. Большинство компилируемых языков программирования требуют явного объявления типов данных переменных. Это позволяет более эффективно использовать ресурсы компьютера и на ранних этапах разработки обнаруживать ошибки типизации, что способствует повышению надежности и безопасности программ.

Разработка программ на компилируемых языках может быть более трудоемкой, поскольку такие языки обычно требуют более строгого синтаксиса и языковых конструкций. Ошибка в исходном коде может привести к сложно обнаружимым ошибкам компиляции, что требует дополнительного времени на отладку и исправление проблемы. Компилируемые языки программирования требуют повторной компиляции программного кода при каждом изменении, что может быть трудоемким и затратным процессом при работе с большими проектами. Это приводит к снижению гибкости и скорости разработки программ.

К современным компилируемым языкам программирования относятся C, C++, Rust, Go.

Интерпретация – это процесс выполнения исходного кода программы путем его анализа и последовательной интерпретации программой интерпретатором во время исполнения ПО. В отличие от компилированных языков программирования, где исходный код предварительно преобразуется в процессорные инструкции и сохраняется в виде исполняемого файла, при интерпретации программы каждая инструкция анализируется и выполняется непосредственно в процессе работы программы. Языки программирования, которые используют интерпретацию как способ трансляции исходного кода в процессорные инструкции называют интерпретируемыми языками.

Одним из основных преимуществ интерпретации программ является ее платформенная независимость. Интерпретируемые языки программирования могут быть выполнены на различных платформах без необходимости перекомпиляции кода. При интерпретации программы можно легко отслеживать и исправлять ошибки, так как интерпретатор выполняет код построчно. Интерпретируемые языки обычно поддерживают динамическую типизацию, что означает, что типы переменных могут быть определены во время выполнения программы.

Однако, интерпретация программ также имеет свои минусы. Один из основных недостатков – это низкая производительность по сравнению с компилируемыми языками программирования. Поскольку интерпретатор выполняет программу на лету, время выполнения может быть дольше, чем у программы, скомпилированной в машинный код. Это особенно заметно при обработке больших объемов данных или выполнении тяжелых вычислений.

К современным интерпретируемым языкам программирования относятся Python, JavaScript, PHP, Ruby, SQL.

Существуют также байт-код языки программирования, которые используют как компиляцию, так и интерпретацию. Байт-код языки программирования используют байт-код, чтобы представлять исходный код программы в промежуточной форме. Компилятор преобразует исходный код в последовательность инструкций, которые являются низкоуровневыми и промежуточными для выполнения на виртуальной машине или интерпретаторе. Этот промежуточный код называется байт-кодом, поскольку он представлен в виде последовательности байтов.

Одним из основных преимуществ использования байт-кода является платформенная независимость. Построение виртуальной машины или интерпретатора, который выполняет байт-код, позволяет запускать программы на разных платформах без изменения исходного кода. Байт-код также часто используется для улучшения производительности программ. Он может быть скомпилирован наиболее эффективным образом для конкретной виртуальной машины или интерпретатора, что приводит к более быстрому выполнению программы.

К современным байт-код языкам программирования относятся Java, C#, Kotlin и Scala (работают поверх Java).

На рисунке 8 представлена ментальная карта классификации языков программирования по критерию преобразования исходного кода в процессорные инструкции.

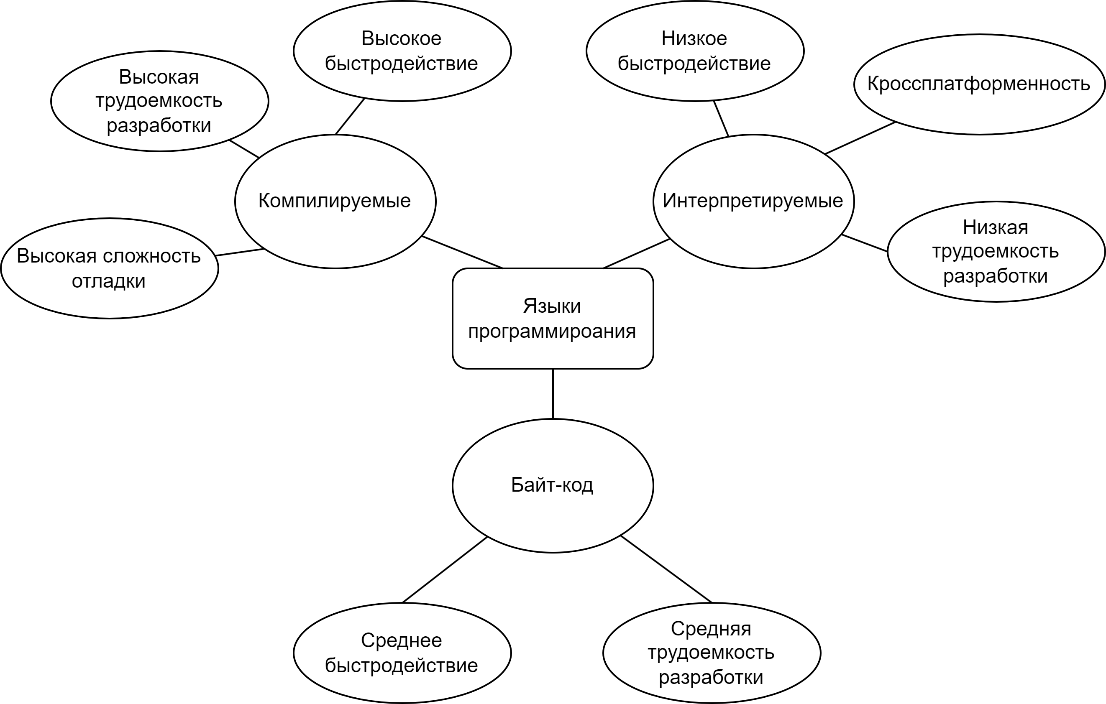


Рисунок 8 – Ментальная карта классификации языков программирования

**2. Анализ исследуемых данных**

На основе проведенного анализа языков программирования необходимо выбрать из множества существующих те, которые потенциально обладают характеристиками быстродействия и эффективности разработки с использованием их. А также необходимо произвести анализ алгоритмов, которые используются в качестве тестируемого

**2.1 Анализ исследуемых языков программирования**

Для выбора языков программирования для исследования составим критерии, по которым будет произведена оценка:

- скорость – насколько быстро язык программирования исполняет заданный алгоритм;

- актуальность – насколько язык современен и какой он имеет потенциал в будущем для дальнейшего перспективного развития эмулятора как программного продукта;

- распространенность – популярность языка среди сообщества разработчиков, и как следствие количество доступной информации и документации в сети Интернет;

- количество интегрированных абстракций и синтаксиса – набор конструкций, абстракций, парадигм и специальных средств, которые повышают эффективность разработки с использованием языка программирования;

- экосистема – наличие интегрированных сред разработки (IDE) с автоматизацией рутинных процессов, наличие компилятора или интерпретатора без программных ошибок, наличие эффективной системы отладки разрабатываемого программного продукта.

Проведем обзор и анализ выбранных языков программирования для исследования их быстродействия:

**Python** – интерпретируемый скриптовый язык программирования, разработанный в конце 1980-х годов Гвидо ван Россумом. Он обладает простым синтаксисом и является самым популярным языком программирования на данный момент по версии индекса TIOBE [13]. Одним из ключевых преимуществ Python является его простота использования. Python предлагает обширные встроенные абстракции, что делает разработку на нем высокоэффективной. Python является интерпретируемым языком, и как следствие он гораздо медленнее по сравнению с компилируемыми языками, но существуют разные способы оптимизации по критерию быстродействия. Python известен своим множеством библиотек и фреймворков, которые делают его очень эффективным инструментом для решения широкого спекта задач.

**Cython** – язык программирования, который представляет собой надстройку Python с возможностью явного определения типов данных. Он позволяет разработчикам писать высокопроизводительный код, близкий к языку C, который можно легко интегрировать с существующим кодом Python. Одним из главных преимуществ Cython является его производительность. Благодаря явному определению типов данных и использованию статической типизации, Cython может синтезировать оптимизированные по критерию быстродействия инструкции процессора. Cython активно используется в проектах, где требуется компромисс между простотой разработки и производительностью. Cython не так распространен, как Python, но все же имеет свою аудиторию и растет в популярности. Cython предоставляет ту же интеграцию и синтаксис, что и Python, добавляя возможность использования статической типизации.

**JavaScript** – язык программирования, разработанный Netscape Communications Corporation в 1995 году. Он широко используется для создания интерактивных веб-страниц и веб-приложений, и наряду с HTML и CSS, является одним из основных языков для веб-разработки. JavaScript также может быть использован для разработки серверной части веб-приложений с использованием платформы Node.js. Это позволяет разрабатывать полноценное ПО, которое может работать как на стороне клиента, так и на стороне сервера.

JavaScript является интерпретируемым языком, но множество встроенных оптимизаций повышают его быстродействие. JavaScript является одним из самых распространенных языков программирования и имеет огромную аудиторию разработчиков. JavaScript предоставляет множество интегрированных абстракций и имеет легкий для понимания синтаксис.

**C#** – был разработан Microsoft в 2000 году и является ключевым языком для платформы «.NET». C# сочетает в себе сильные стороны C++ и удобство Java, что делает его очень популярным среди разработчиков. Одним из основных преимуществ C# является его объектно-ориентированность. Он предоставляет механизмы наследования, инкапсуляции и полиморфизма для построения модульных и расширяемых программ. Кроме того, C# имеет встроенные средства для автоматического управления памятью, что облегчает разработку и устраняет множество ошибок, связанных с неправильным использованием памяти.

C# медленнее относительно компилируемых языков, но современные решения по оптимизации по критерию быстродействия делают его довольно эффективным. C# применяется для разработки приложений под ОС Windows и игр на платформе Unity. C# предлагает широкий набор интегрированных абстракций и имеет синтаксис, являющийся комбинацией C++ и Java. C# сопровождается обширной экосистемой, включая интегрированную среду разработки Visual Studio и множество библиотек для различных целей.

**Kotlin** – язык программирования, разработанный JetBrains в 2011 году, специально для разработки приложений на платформе Java. Kotlin является статически типизированным языком, который обладает простым и понятным синтаксисом и представляет собой расширение возможностей Java. Одним из основных преимуществ Kotlin является его совместимость с Java. Это позволяет разработчикам легко интегрировать Kotlin в существующие проекты на Java и переиспользовать код. Он имеет множество усовершенствований по сравнению с Java. Кроме того, Kotlin предоставляет широкие возможности синтаксиса, что упрощает написание безопасного и эффективного кода.

Является байт-код языком, компилируется в Java байт-код, и проявляет хорошую производительность, но все ещё недостающую до уровня компилируемых языков. Kotlin не так широко распространен, как Java, но его популярность стремительно растет.

**Rust** – относительно новый системный язык программирования, разработанный Mozilla Research в 2010 году. Прежде всего, Rust известен своими возможностями по повышению безопасности и предотвращению множества типичных ошибок программирования. Одним из ключевых преимуществ Rust является его система контроля доступа к памяти, которая позволяет предотвратить ошибки, связанные с несогласованным доступом к памяти. Это делает Rust особенно безопасным для разработки системного ПО, где низкоуровневые ошибки могут привести к серьезным проблемам.

Rust также обладает эффективной системой типов и статическим анализом, что позволяет выявлять ошибки во время компиляции. Это сильно упрощает отладку и повышает качество разрабатываемого кода. Кроме того, Rust предоставляет механизм автоматического управления памятью. Rust является компилируемым языком программирования, который показывает отличную производительность. Rust все еще развивается и не так широко распространен, как другие языки, но потенциально может найти применение в специализированных областях. Имеет развивающуюся экосистему с библиотеками и инструментами для разработки.

**C** – разработанный в начале 1970-х годов язык программирования, является одним из наиболее широко используемых языков в индустрии разработки ПО. Он известен своими возможностями и эффективностью, а также широкой поддержкой различных архитектур и операционных систем. Одним из главных преимуществ C является его низкий уровень абстракции от аппаратной части, что позволяет разработчикам иметь полный контроль над памятью и процессором. Это делает C особенно подходящим для разработки системного и встроенного ПО, операционных систем и драйверов устройств.

В данном исследовании C используется как эталон языка программирования с наибольшим быстродействием и распространенностью. Таким образом для разработки эмулятора встраиваемых он не рассматривается в связи с отсутствием поддержки объектно-ориентированного программирования и малым количеством интегрированных абстракций.

**2.2 Анализ исследуемых алгоритмов**

Для проведения исследования быстродействия языков программирования были выбраны 3 алгоритма. Операции, выполняемые в них, являются самыми распространенными операциями, которые выполняет процессор. Следовательно, эти операции будут выполнятся чаще всего в эмуляторе встраиваемых систем. Необходимо, чтобы язык программирования, на котором будет реализован эмулятор максимально быстро мог выполнять именно эти операции. В приведенных алгоритмах не рассматриваются вспомогательные операции, такие как выделение памяти, чтение данных, вывод результатов.

Рассмотрим выбранные алгоритмы по подробнее.

**Алгоритм перемножения матриц** — это процесс, который состоит в умножении двух матриц для получения третьей матрицы, называемой матрицей произведения. Алгоритм реализован с использованием циклов, где каждому элементу матрицы произведения ставится в соответствие сумма произведений элементов соответствующих строк и столбцов. Отличие в реализации заключается в том, что вторая матрица транспонируется для более эффективного использования памяти. Алгоритм перемножения матриц имеет сложность , где – размер стороны матрицы так как содержит три вложенных цикла. Временная сложность этого алгоритма зависит от размеров входных матриц и может быть существенной для больших матриц.

Блок схема алгоритма перемножения матриц представлена на рисунке 9.

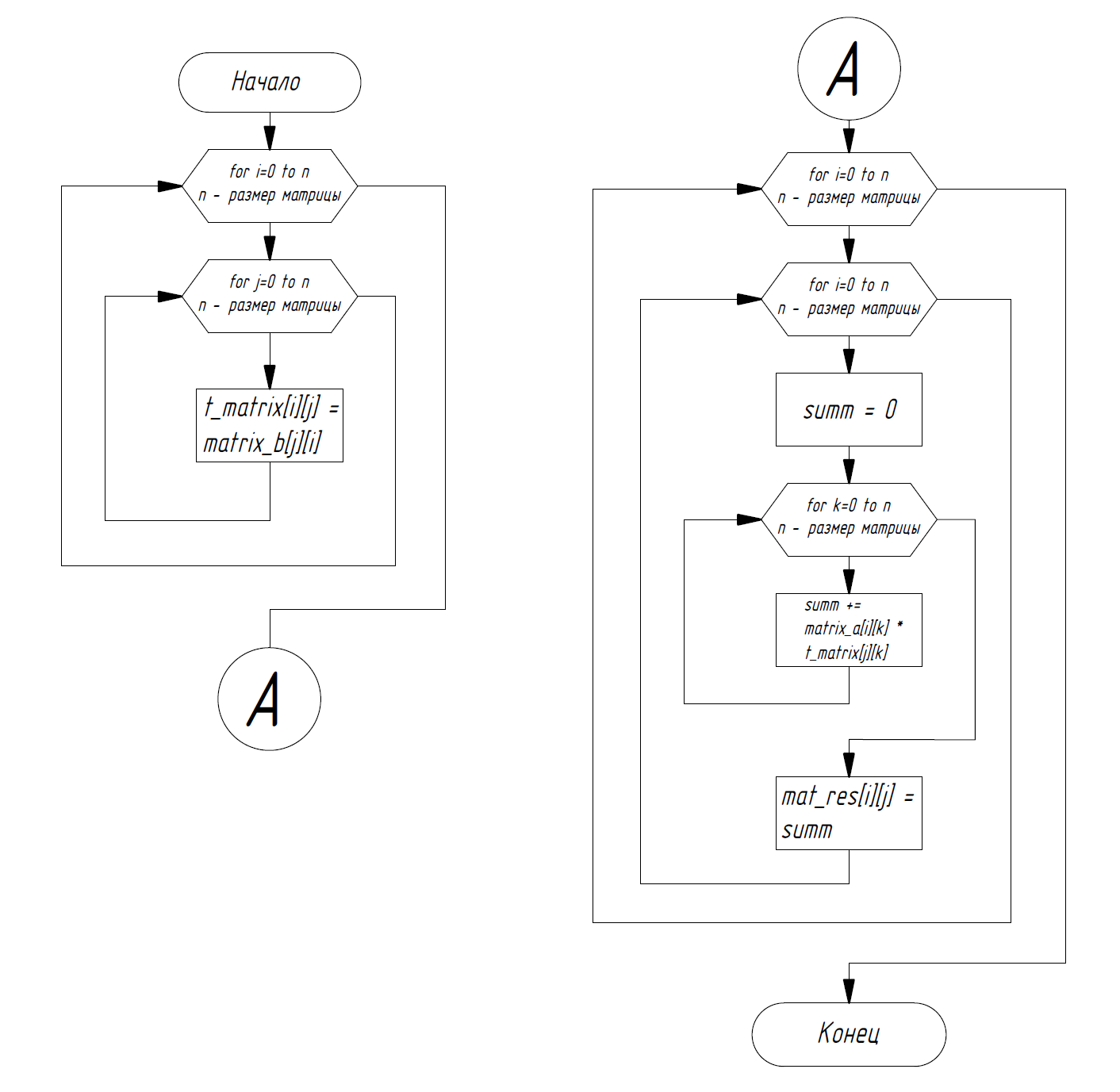


Рисунок 9 – Блок схема алгоритма перемножения матриц

**Быстрая сортировка** – представляет собой один из наиболее эффективных методов сортировки массивов данных. Алгоритм быстрой сортировки реализуется путем выбора опорного элемента из исходного массива и последующего разделения массива на две подгруппы: элементы, меньшие опорного, и элементы, большие опорного. Эти подгруппы рекурсивно подвергаются процессу сортировки с использованием быстрой сортировки, пока не будет достигнуто конечное упорядочивание. Этот процесс называется "разбиением".

Алгоритм быстрой сортировки обладает высокой эффективностью, в среднем имеет временную сложность , где - количество элементов в массиве. Однако, в худшем случае, его временная сложность может достигать , что происходит при неудачном выборе опорного элемента или в случае, когда массив уже упорядочен по возрастанию или убыванию.

Блок схема алгоритма быстрой сортировки представлена на рисунке 10.

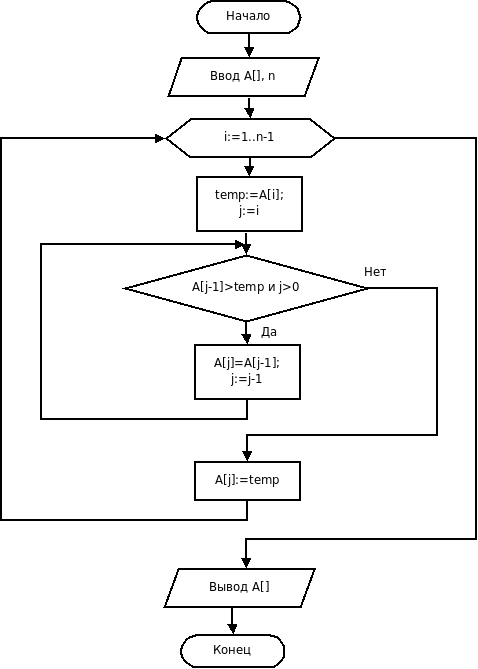


Рисунок 10 – Блок схема алгоритма быстрая сортировка

**Программный алгоритм решета Аткина** является эффективным и быстрым методом для нахождения простых чисел в заданном числовом интервале. Этот алгоритм получил свое название в честь его автора, перуанского математика Дж.Аткина, и имеет широкое применение в области криптографии, анализа данных и других прикладных задач.

Решето Аткина основывается на идее использования квадратных шаблонов для определения простоты чисел в заданном интервале. В отличие от других алгоритмов, которые используют деление на простые числа, решето Аткина опирается на решение квадратных сравнений.

Процесс алгоритма начинается с инициализации специального булевого массива размером , где - верхняя граница интервала, в котором требуется найти простые числа. Затем, начиная с трех, происходит перебор чисел из заданного интервала. При нахождении числа , для которого выполняется одно из квадратных сравнений, его мультипликативный вес в булевом массиве обновляется.

После завершения прохода по всем числам, в булевом массиве отмечаются индексы, соответствующие простым числам. Таким образом, по окончании работы алгоритма получается подмножество простых чисел в заданном интервале.

Блок схема алгоритма решето Аткина представлена на рисунке 11.

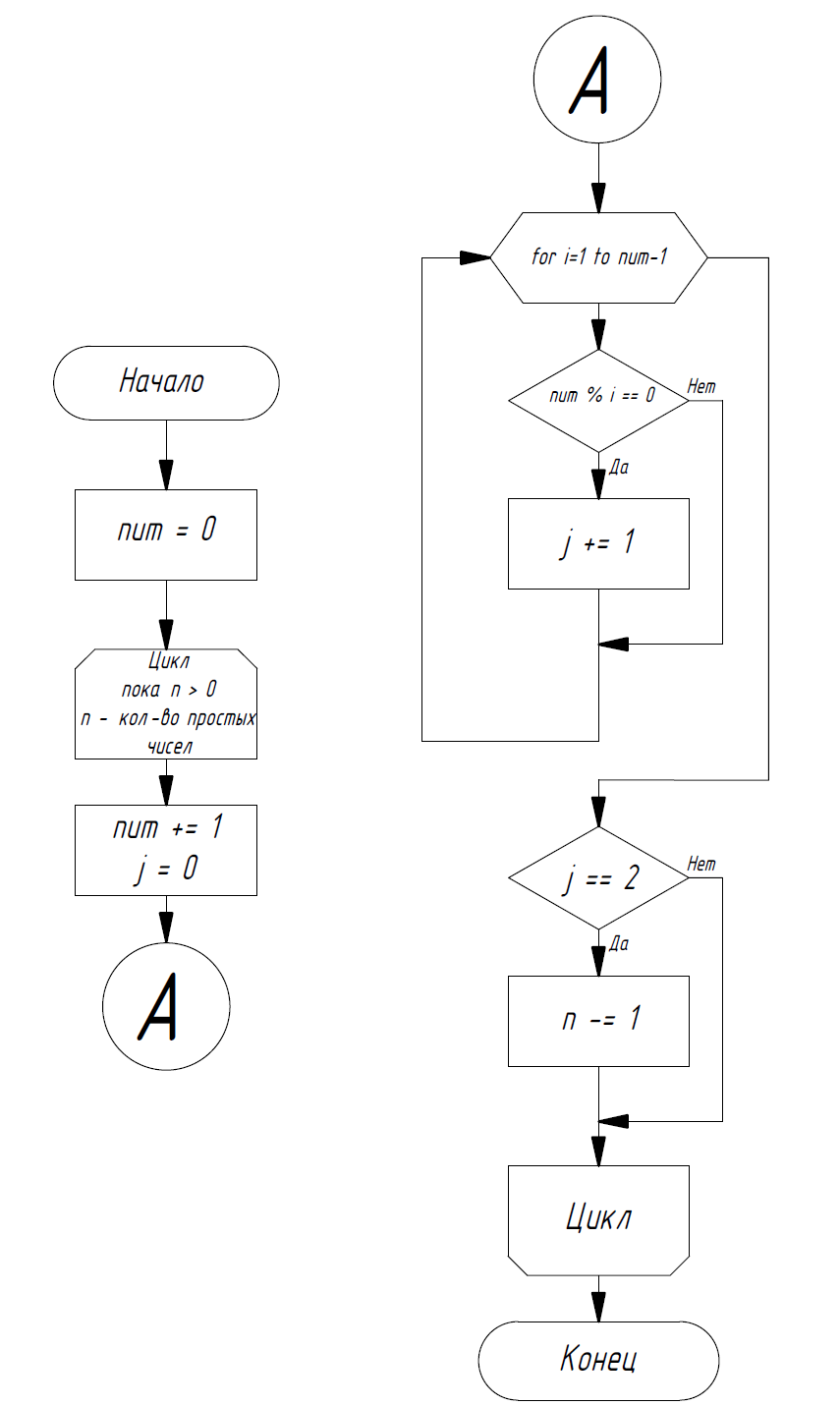


Рисунок 11 – Блок схема алгоритма решето Аткина

**3. Результаты тестирования и исследования**

В данной главе представлены результаты тестирования и исследований, направленных на оценку быстродействия языков программирования по выполнению описанных алгоритмов. Исследование было проведено с использованием разработанного программного средства.

**3.1 Описание методологии исследования**

Методология исследований была разработана с целью создания стабильного окружения, в котором можно было бы оценить влияние языка программирования на скорость исполнения алгоритмов. В качестве основы использовались стандартные сценарии тестирования языков программирования на быстродействие.

Программное средство для проведения тестирования установлено на виртуальную машину с ОС семейства Linux Ubuntu 22.04. Параметры виртуальной машины:

- Процессор AMD Ryzen 5 5600U with Radeon Graphics 2 ядра;

- Оперативная память 4 Гб;

- Постоянная память 60 Гб;

- Видео карта не используется.

Тестирование проводится в Docker контейнере, созданном на основе ОС семейства Linux Ubuntu 22.04. Он позволяет поддерживать окружение в одном и том же стабильном состоянии, что позволяет снизить влияние различных процессов, происходящих на вычислительной системе, на которой происходит тестирование.

В программном средстве для тестирования предусмотрена конфигурация последовательности тестов на определенных языках программирования и определенных алгоритмов. Таким образом, созданы 3 конфигурации на каждый из представленных алгоритмов. Программное средство для тестирования также имеет функциональность запуска нескольких циклов одних и тех же тестов. Это позволяет провести множество одинаковых тестов и посчитать среднее арифметическое полученных результатов. Для каждого цикла тестов проводится генерация нового набора входных данных. Тестирование проводится при 100 циклах повторов тестов. Исходя из статистических методов данный метод тестирование покажет результат, наиболее близкий к реальности.

Для алгоритма решето Аткина программное средство генерирует числа в диапазоне от 10000 до 15000. Для алгоритма перемножения матриц генерируются матрицы размером 2000 на 2000 элементов с числами в диапазоне от 0 до 65536. Для алгоритма быстрой сортировки генерируются массивы данных размером 1000000 элементов с числами в диапазоне от 0 до 4294967294.

В процессе проведения тестирования происходит журналирование всех событий, происходящий в процессе. Это необходимо для повышения эффективности отладки в случае, если ошибка произойдет в одном из множества тестов, исполняемых подряд.

В результате проведения тестирования будет создана столбчатая диаграмма, на вертикальной оси которой будет отображено время в секундах, а на горизонтальной оси представлены языки программирования.

**3.2 Представление результатов тестирования и их анализ**

Результаты тестирования были подвергнуты анализу с целью выявления основных тенденций и зависимостей. Получившиеся результаты оказались не столь однозначными, как можно было предположить. На рисунках 12-14 представлены результаты проведенного исследования.

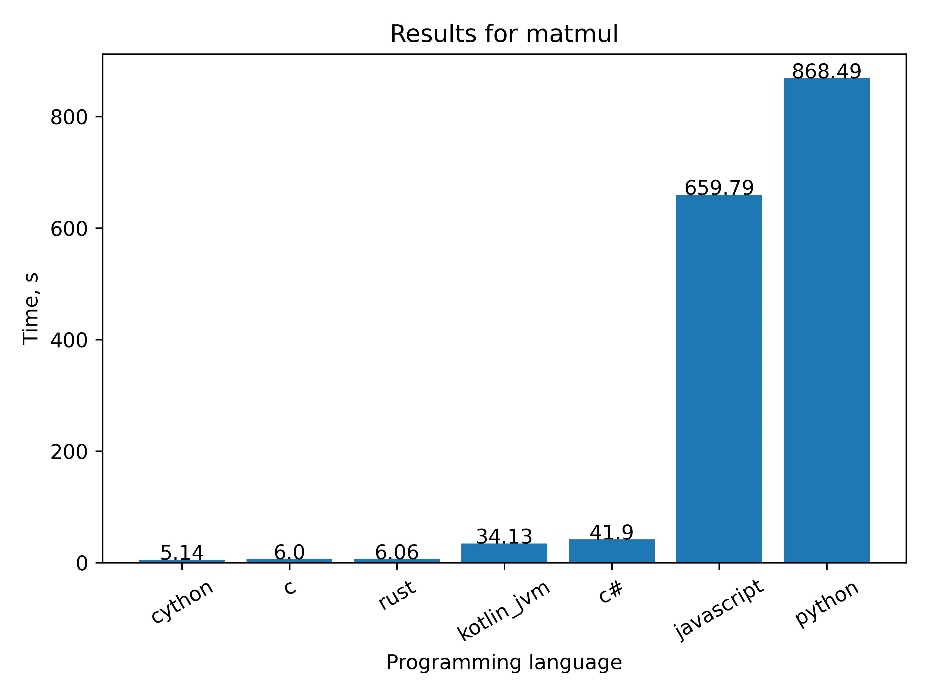


Рисунок 12 – Результаты тестирования алгоритма умножения матриц

На основании полученных результатов были подтверждены утверждения о компилируемых, интерпретируемых байт-код языках программирования. Быстрее всего перемножение матриц выполняют компилируемые языки, а интерпретируемые показывают худшие показатели. Байт-код языки занимают промежуточное положение между ними. Неожиданно высокий результат показал Cython, обойдя по скорости даже C.

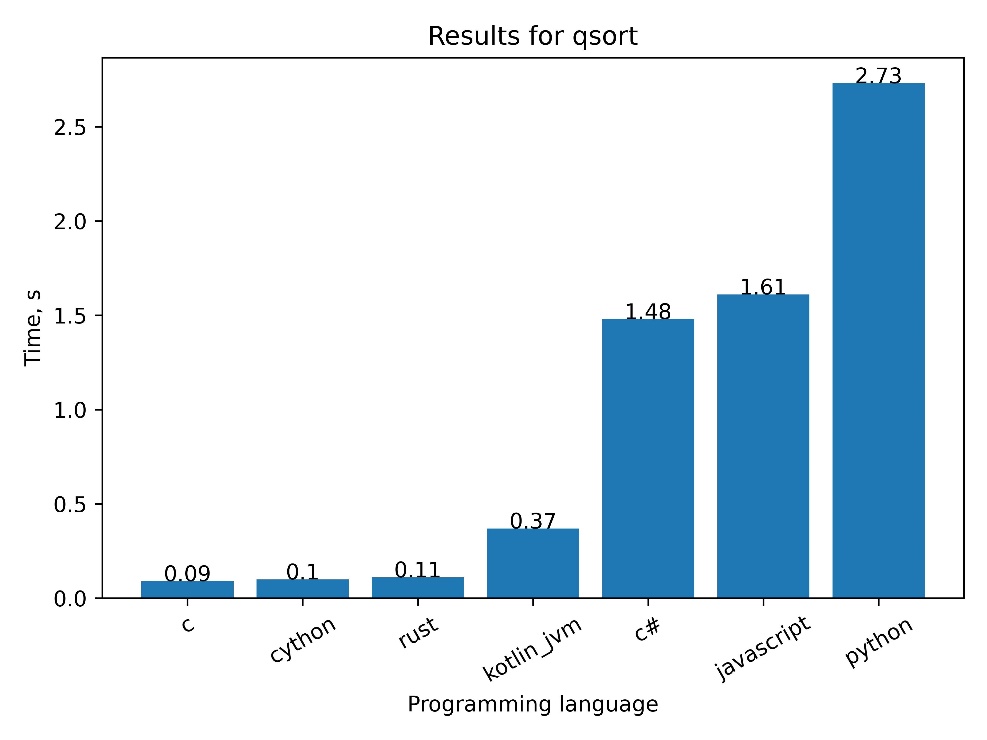


Рисунок 13 – Результаты тестирования алгоритма быстрой сортировки

Результаты данного тестирования показывают схожие результаты с тестированием скорости умножения матриц, однако язык C показал в данном случае наилучший результат, который и был от него ожидаем.

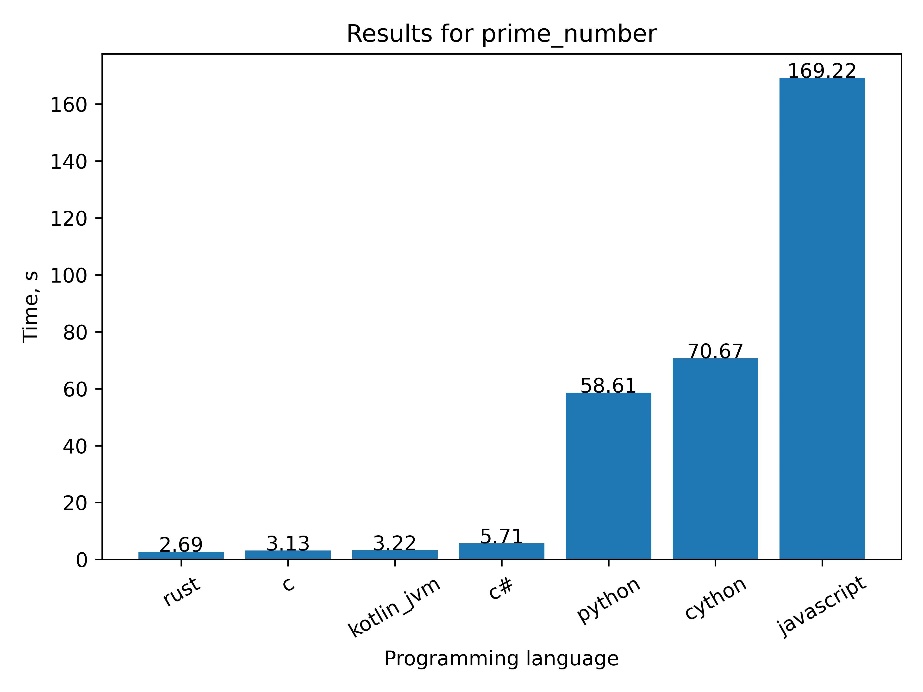


Рисунок 14 – Результаты тестирования алгоритма решето Аткина

В данном тестировании наблюдается существенное отличие от предыдущих. JavaScript почти в 3 раза оказался медленнее, чем Python. В свою очередь Cython оказался также медленнее чем Python. Rust же показал наилучший результат среди всех тестов.

Анализ результатов позволил выявить, что для различных алгоритмов различные языки программирования показывают разные результаты. Закономерность все ещё присутствует, но она не так явно представлена на данных результатах. Таким образом, можно сделать вывод о том, что какие-то составляющие языка программирования работают лучше, чем другие

**3.3 Выводы относительно результатов тестирования**

Из результатов исследования следует, что скорость работы языков программирования может значительно различаться в зависимости от их особенностей и оптимизаций. Например, языки программирования с компилируемым кодом, такие как C или Rust, демонстрировали более высокую производительность по сравнению с языками с интерпретируемым кодом, например Python или JavaScript. Однако, следует отметить, что скорость работы не является единственным критерием при выборе языка программирования, учитывать также следует удобство использования, доступность разработчиков и другие факторы.

На основании результатов данного исследования было выявлено, что выбор языка программирования должен быть осуществлен с учетом конкретных требований и целей проекта. Однако, необходимо отметить, что результаты данного исследования являются относительными и могут зависеть от множества факторов, включая конкретную реализацию языка программирования, оптимизацию кода, аппаратное обеспечение и другие условия.

**Обсуждение**

Результаты наших исследований внесли значительный вклад в понимание влияния языка программирования на скорость исполнения определенных алгоритмов на вычислительной системе. В данной главе мы проведем анализ полученных результатов, рассмотрим их в контексте существующих знаний и технологических трендов, а также обсудим возможные практические применения и дальнейшие направления исследований.

Интересным аспектом наших результатов является подтверждение того, что компилируемые языки программирования имеют характеристику быстродействия на порядки выше, чем у интерпретируемых языков. Таким образом, целесообразность разработки эмулятора встраиваемых систем с использованием интерпретируемых языков программирования ставится под сомнение. Но также стоит отметить, что в данном исследовании никак не была учтена или специально использована JIT-компиляция. Она представляет собой эффективный инструмент по увлечению быстродействия интерпретируемых и байт-код языков программирования.

Примечательно, что на различных алгоритмах некоторые языки программирования показали кардинально отличающиеся результаты. Это открывает перспективы для новых исследований с использованием большего числа алгоритмов, которые позволят выявить более точные закономерности в понимании их быстродействия.

В целом, проведенные исследования открывают новые перспективы для развития современных эмуляторов встраиваемых систем и разработки новых методов увеличения их быстродействия.

**Заключение**

В данной статье мы провели обширное исследование, направленное на анализ быстродействия языков программирования в контексте задачи эмуляции ПО встраиваемых систем. Полученные результаты подчеркнули важность выбора языка программирования при разработке эмулятора встраиваемых систем. В заключение мы хотели бы подытожить ключевые моменты и выделить перспективы для будущих исследований.

В данной работе было проведено исследование характеристик быстродействия современных языков программирования при выполнении операций и алгоритмов, часто используемых при эмуляции встраиваемых систем. Было разработано программное средство, позволяющее оценить производительность различных языков программирования при выполнении заданных алгоритмов. Предложены рекомендации по использованию средств разработки ПО для создания эффективного эмулятора встраиваемых систем.

Результаты исследования могут быть полезны для инженеров и разработчиков, работающих в области проектирования встраиваемых систем и разработки программного обеспечения под них. Получившиеся рекомендации помогут повысить эффективность разработки и тестирования ПО для встраиваемых систем, а также снизить возможные риски и проблемы, связанные с некорректным поведением эмулятора.

В дальнейшем, основываясь на результатах данной работы, можно провести дополнительные исследования в области оптимизации эмуляторов встраиваемых систем и разработки новых средств разработки ПО, учитывающих особенности работы с данными системами.

В целом, представленная работа является важным вкладом в область разработки эмуляторов встраиваемых систем и может стать полезным справочным материалом для специалистов, занимающихся данной проблематикой.

**Литература**

1. Петров Алексей Александрович, Кольманович Антон Сергеевич Встраиваемые системы // Научные исследования. 2018. №5 (24).
2. Zobov O.V., Vlasov A.I., Juravleva L.V., The building data collection devices problems analysis in the industrial iot paradigm // III International conference on advanced technologies in materials science, mechanical and automation engineering, 2021.
3. Ефимов С.Н., Терсков В.А., Галушин П.В., Ярков К.В. Модель надежности многопроцессорных аппаратно-программных комплексов систем управления реального времени с мультиверсионным программным обеспечением // Вестник МГТУ им. Н.Э. Баумана. Серия «Приборостроение». 2021. №4 (137).
4. Заборовский Никита Владимирович, Тормасов Александр Геннадьевич Моделирование многопоточного исполнения программы и метод статического анализа кода на предмет состояний гонки // Прикладная информатика. 2011. №4 (34).
5. Peter Boonstoppel, Cristian Cadar, Dawson Engler, RWset: Attacking Path Explosion in Constraint-Based Test Generation // Lecture Notes in Computer Science book series (LNTCS, volume 4963)
6. 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, 2022, 42, (3):e97363.
7. Сельвесюк Н.И., Островский А.С., Русанов П.В., Комахин М.О., Объектно-ориентированное проектирование системы эмуляции программного интерфейса приложения в задачах обеспечения информационной безопасности // Информатика и системы управления, 2 (56), 2018.
8. B. Zhang et al. A Survey on I/O Virtualization and Optimization. 2010 Fifth Annual ChinaGrid Conference, Guangzhou, China, 2010, pp. 117-123.
9. Dileep P., Devesh G., Raghavendra R., et al, Verification of Linux device drivers using device virtualization // 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 2015, pp. 694-698.
10. Узеньков Д.А., Власов А.И., Гладких А.А., др. Динамическая модификация внутреннего программного обеспечения встраиваемых устройств для решения задач обратной разработки // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). Выпуск 2. 2021. С. 63-69.
11. S. Gill, The Diagnosis of Mistakes in Programmes on the EDSAC // Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences, Vol. 206, No. 1087, May 1951.
12. Jakob Engblom, Guillaume Girard, Bengt Werner, Testing Embedded Software using Simulated Hardware // ERTS 2006.
13. Индекс TIOBE, URL: https://www.tiobe.com/tiobe-index/ (дата обращения: 10.21.2023).
14. Stefan Marr, Benoit Daloze, and Hanspeter Mössenböck, Cross-language compiler benchmarking: are we fast yet? In Proceedings of the 12th Symposium on Dynamic Languages (DLS 2016). Association for Computing Machinery, New York, NY, USA, 120–131.
15. Иванов Сергей Олегович, Ильин Дмитрий Владимирович, Большаков Иван Юрьевич Сравнительное тестирование языков программирования // Вестник ЧГУ. 2017. №3.
16. Коровин И.В., Пулькин И.А., Веранян А.С. Исследование скоростей выполнения базовых математических задач популярных языков программирования // Экономика и качество систем связи. 2019. №3 (13).

**References**

1. Petrov Alexey Alexandrovich, Kolmanovich Anton Sergeevich Embedded systems // Scientific research. 2018. No. 5 (24).

2. Zobov O.V., Vlasov A.I., Juravleva L.V., The building data collection devices problems analysis in the industrial iot paradigm // III International conference on advanced technologies in materials science, mechanical and automation engineering, 2021.

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 multiversion software // Vestnik MSTU im. N.E. Bauman. Series "Instrument making". 2021. No. 4 (137).

4. Zaborovsky Nikita Vladimirovich, Tormasov Alexander Gennadievich Modeling of multi-threaded program execution and a method of static code analysis for race conditions // Applied Informatics. 2011. No. 4 (34).

5. Peter Boonstoppel, Cristian Cadar, Dawson Engler, RWset: Attacking Path Explosion in Constraint-Based Test Generation // Lecture Notes in Computer Science book series (LNTCS, volume 4963)

6. 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, 2022, 42, (3):e97363.

7. Selvesyuk N.I., Ostrovsky A.S., Rusanov P.V., Komakhin M.O., Object-oriented design of an application software interface emulation system in information security tasks // Informatics and Control Systems, 2 (56 ), 2018.

8. B. Zhang et al. A Survey on I/O Virtualization and Optimization. 2010 Fifth Annual ChinaGrid Conference, Guangzhou, China, 2010, pp. 117-123.

9. Dileep P., Devesh G., Raghavendra R., et al, Verification of Linux device drivers using device virtualization // 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 2015, pp. 694-698.

10. Uzenkov D.A., Vlasov A.I., Gladkikh A.A., etc. Dynamic modification of internal software of embedded devices for solving reverse engineering problems // Problems of development of advanced micro- and nanoelectronic systems (MES). Issue 2. 2021. pp. 63-69.

11. S. Gill, The Diagnosis of Mistakes in Programs on the EDSAC // Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences, Vol. 206, No. 1087, May 1951.

12. Jakob Engblom, Guillaume Girard, Bengt Werner, Testing Embedded Software using Simulated Hardware // ERTS 2006.

13. TIOBE Index, URL: https://www.tiobe.com/tiobe-index/ (access date: 21/10/2023).

14. Stefan Marr, Benoit Daloze, and Hanspeter Mössenböck, Cross-language compiler benchmarking: are we fast yet? In Proceedings of the 12th Symposium on Dynamic Languages (DLS 2016). Association for Computing Machinery, New York, NY, USA, 120–131.

15. Ivanov Sergey Olegovich, Ilyin Dmitry Vladimirovich, Bolshakov Ivan Yurievich Comparative testing of programming languages // Bulletin of ChSU. 2017. No. 3.

16. Korovin I.V., Pulkin I.A., Veranyan A.S. Study of the speed of execution of basic mathematical problems in popular programming languages // Economics and quality of communication systems. 2019. No. 3 (13).

**Сведения об авторах**

Гладких Алексей Алексеевич, кандидат технических наук, доцент кафедры ИУ4 «Проектирование и технология производства электронной аппаратуры» МГТУ им. Н.Э. Баумана, Москва, Россия. Email: gladkikhalexei@gmail.com

Кутаев Кирилл Сергеевич, магистр кафедры ИУ4 "Проектирование и технология производства электронной аппаратуры", МГТУ им. Н.Э. Баумана, Москва, Россия. Email: kirSM2010@gmail.com

**Information about authors**

Gladkikh Aleksey Alekseevich, Candidate of Technical Sciences, Associate Professor of the Department of Engineering Unit 4 “Design and Production Technology of Electronic Equipment” MSTU. N.E. Bauman, Moscow, Russia. Email: gladkikhalexei@gmail.com

Kutaev Kirill Sergeevich, Master of the Department of IU4 "Design and Manufacturing Technology of Electronic Equipment", MSTU named after. N.E. Bauman, Moscow, Russia. Email: kirSM2010@gmail.com