Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Национальный исследовательский университет ИТМО»

(Университет ИТМО)

Факультет ПИиКТ

Образовательная программа Веб-технологии

Направление подготовки (специальность) 09.04.04 Программная инженерия

ОТЧЕТ

по научно-исследовательской работе

Тема задания: Анализ зарубежных и отечественных научных источников по теме исследования «Исследование влияния модулей WebAssembly на производительность приложений JavaScript»

Обучающийся Валиуллин А.Р., Р4107

Согласовано:

Руководитель практики от университета: Готская И. Б., профессор, ФПИиКТ

Практика пройдена с оценкой отлично

Дата 27.01.2024

Санкт-Петербург

2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

WebAssembly становится все более популярным благодаря своей способности ускорять работу веб-приложений, особенно в сложных и ресурсоемких приложениях. Популярности WebAssembly также способствует совместимость бинарных модулей в различных исполняемых средах JavaScript, более высокий уровень безопасности, предполагаемый проверками компиляторами WebAssembly.

Актуальность работы обусловлена возросшими требованиями к производительности веб-приложений, необходимостью сохранения совместимости реализуемых функциональных возможностей кода на различных языках. В связи с этим, возникает потребность в проведении сравнительного анализа исполняемых модулей приложения, скомпилированных в бинарный формат WebAssembly. Это обосновывает актуальность выбранной темы научно-исследовательской работы.

Цель НИР: выполнить анализ зарубежных и отечественных источников для формирования представления о возможностях и особенностях применения технологии WebAssembly при разработке приложений.

Объект исследования: информационные системы, использующие технологию WebAssembly для расширения возможностей веб-приложений.

Предмет исследования: анализ отечественной и зарубежной литературы по теме исследования.

Задачи исследования:

1. Составить обзор текущего состояния проблемы производительности веб-приложений, которые используют бинарные модули WebAssembly.
2. Составить обзор российских научных исследований, посвященных вопросам производительности приложений, инструментов оценки характеристик.

3. Составить обзор зарубежных научных исследований.

Теоретические основы исследований. Анализ источников показал, что популярность технологии WebAssembly растет в качестве решения проблемы с производительностью кода, однако не является полной заменой JavaScript.
В качестве теоретической основы выполнения НИР выступили работы авторов, в которых рассматривается анализ производительности приложений с использованием расширений, скомпилированных в бинарные модули WebAssembly (С.К. Кособродов, В.В. Рокотянская, В.С. Абрамов, В. Н. Гридин, В.И. Анисимов, С.А. Васильев, Н.И. Любицкий, К.Р. Мальцев), особенности использования низкоуровневого кода (О.В. Бородин, В.А. Егунов, В.П. Плотников, А.В. Григоренко) и статического исследования использования язык программирования в WebAssembly (A. Hilbig , D. Lehmann, Pradel).

Информационная база исследования: eLibrary, CyberLeninka, Google Scholar, IEEE, Scopus.

Метода исследования: анализ, синтез, обобщение.

Результаты исследования. Проведенный анализ показал, что как российские, так и зарубежные научные исследования широко рассматривают анализ производительности бинарных модулей WebAssembly через призму вычислительных задач, указывают на то, что WebAssembly не может быть полноценной заменой использования JavaScript, но имеет большую перспективу в качестве дополняющей технологии. В результате анализа производительности как отечественные авторы, так и зарубежные отмечают, что выигрыш в производительности приложением приобретается в зависимости от типа задач, выбранного языка программирования и среды выполнения. Все авторы отмечают особенно хорошую работу модулей WebAssembly построенных на языке Rust как с точки зрения производительности, так и безопасности. Зарубежные авторы, так же исследуют безопасность использования WebAssembly.

1 АНАЛИЗ ОТЕЧЕСТВЕННЫХ НАУЧНЫХ ИСТОЧНИКОВ ПО ТЕМЕ ИССЛЕДОВАНИЯ

Несмотря на растущую популярность и технологические новинки в веб-разработке, вопрос быстрого исполнения кода остается критически важным. От скорости работы приложения или сайта зависит пользовательский опыт, а следовательно, и успешность проекта. На скорость исполнения кода могут влиять различные факторы, включая выбор технологий, оптимизацию кода, использование эффективных инструментов и методов. В свете этих обстоятельств, разработчикам важно уделять постоянное внимание улучшению производительности своих приложений и сайтов.

В своей статье О.В. Бородин, В.А. Егунов, В.П. Плотников рассматривают производительность рекурсивных и итеративных функций Фибоначчи, функции перемножения целых чисел, функции перемножения векторов, реализованных на языках C, C++, Rust и скомпилированных в бинарные модули WebAssembly в сравнении с аналогичными реализациями на JavaScript. Авторы пришли к вывод, что в сравнении с JavaScript, получается, что в среднем WASM быстрее, но при этом возможны индивидуальные ситуации, в которых необходимо разбираться детальнее, потому что возможен случай получения производительности не только в несколько раз лучше, но так и значительно хуже. Это может зависеть от используемого браузера, а также его версии [1].

В научно-практической работе А.В. Григоренко выполнил сравнение производительности фильтров изображений на JavaScript и C++ [2]. Замеры скоростей с применением пяти фильтров представлены на рисунке 1. В этом случае автор показывает, что бинарные модули WebAssembly не могут работать быстрее чем JavaScript, указывая на то, что современные движки браузеров выполняют качественную оптимизацию кода JavaScript, которая

позволяет достичь скорости не хуже скоростей бинарных модулей WebAssembly.

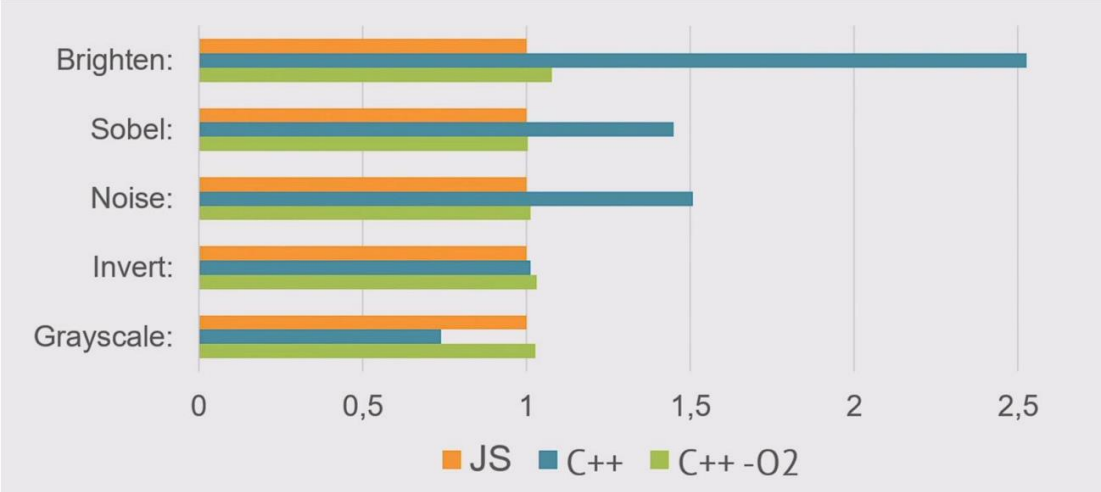

Рисунок 1 - Сравнение производительности C++ с JavaScript в WebAssembly

Авторы В.Н. Гридин, В.И. Анисимов, С.А. Васильев обращают внимание, что [3]:

Для поддержания коммерческой конкурентоспособности современных веб-приложений следует обеспечить высокую степень интерактивности их интерфейса на уровне нативного приложения. Чтобы обеспечить такой уровень быстродействия необходимо иметь эффективно организованный распределенный state management для снижения нагрузки на клиентское приложение. Для обеспечения своевременной поставки необходимой информации, по мере появления данных, без избыточности следует использовать полнодуплексные или комбинированные подходы для установления каналов связи. Сопряжение указанных подходов с интеллектуальным механизмом декларирования структуры данных и методов доступа к ним, предоставляющих возможность установления подписки на необходимые наборы данных, а также формирование графо-подобных структур неопределенного уровня вложенности, позволяют добиться высоко уровня производительности обмена данными. Для обеспечения

эффективности работы с информацией приведенный middleware должен предусматривать работу с колаборативными масштабируемыми системами хранения слабосвязанных данных, использующих в качестве структурных единиц объектную модель, а в качестве носителя информации оперативное запоминающее устройство.

Авторы Д.О. Еремин, Е.И. Холмогорова, отмечают возможность использования языка программирования C# для формирования бинарных модулей WebAssembly на стороне клиента посредством фреймворка Blazor и его высокую производительность сопоставимую с нативной и портируемость такого кода [4].

Автор С.К. Кособродов провел исследование, котором сравнил производительность AssemblyScript и JavaScript на задачах поиска в ширину, быстрого преобразования Фурье, LU-разложения, PageRank, разреженное матрично-векторное умножение [5]. На рисунке 2 представлены результаты замеров времени, выполненные автором.

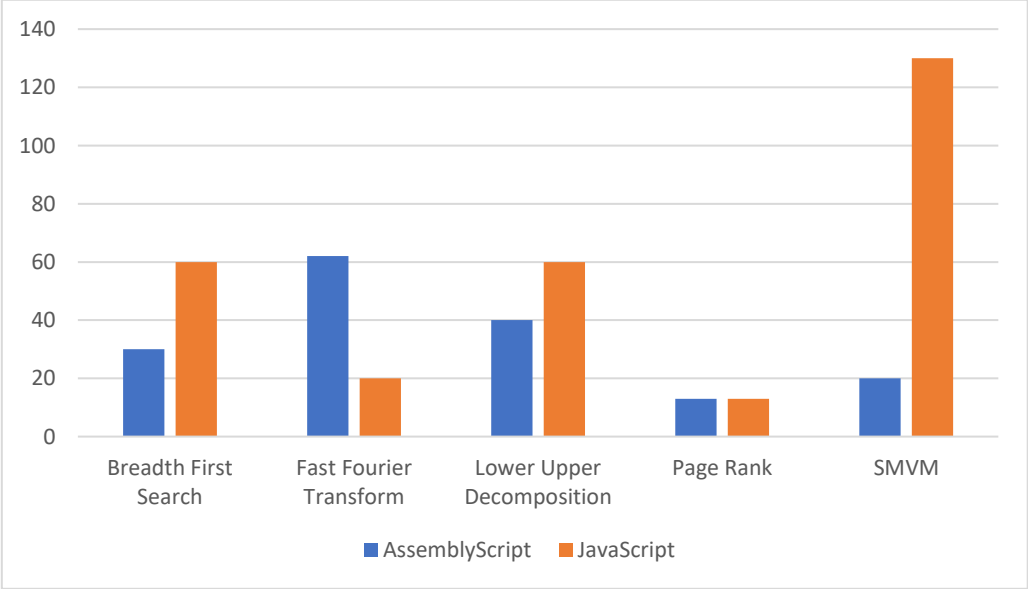

Рисунок 2 - Среднее время выполнения тестов AssemblyScript и JavaScript

А.А. Лымарев исследовал возможность использования технологии React Native и WebAssembly для разработки кроссплатформенных мобильных

приложений. На основании анализа и экспериментов с технологиями выявлено, что в текущий момент использование данного стека для разработки затруднительно, а также предложил использовать специальные полифилы, которые способны добавить возможность использования WebAssembly [6].

Авторы Н.И. Любицкий, К.Р. Мальцев выполнили реализацию алгоритма построения диаграмм Вороного с помощью триангуляции Делоне на языках программирования JavaScript, C++, Rust, Go и пришли к выводу [7]:

На основе полученных данных можно сделать вывод о том, что если работа алгоритма будет выполнятся в настольной версии программы или на серверной части веб-приложения, то для получения самой высокой производительности алгоритма нужно выбирать язык программирования Rust. Разница в производительности алгоритма на языках JavaScript и C++ составляет 10% в пользу C++ [7]. Результаты замеров, выполненные авторами представлены в таблице 1.

Таблица 1 – Время выполнения алгоритма (мс)

Кол-во точек	JavaScript	C++	Rust	Go
100	0,01	0,01	0,02	0,04
1000	0,56	0,50	0,03	0,49
10000	68,20	62,00	3,75	5,55
100000	87,79	79,80	63,63	79,90
1000000	1092,31	993,00	898,78	1272,00
10000000	16767,30	15243,00	11857,00	23481,00

З.А. Овчаров рассмотрел использование WebAssembly в качестве метода решения проблемы замедления работы «толстого» клиента из-за однопоточности вычислительного ядра стандартных браузеров и пришел к выводу, что использование WebAssembly показывает высокую

производительность, при этом вычисления с помощью WebAssembly никак не блокирует основной поток работы браузера [8].

В.В. Рокотянская, В.С. Абрамов провели сравнение скорости выполнения функций перемножения, быстрой сортировки, суммирования, перемножения векторов, вычисления числа Фибоначчи. Результаты их эксперимента приведены на рисунке 3. Авторы отмечают, что [9]:

Технология WebAssembly не является полной заменой JavaScript, а лучше подходит как инструмент, которому следует отдавать в работу большие и трудные вычисления.

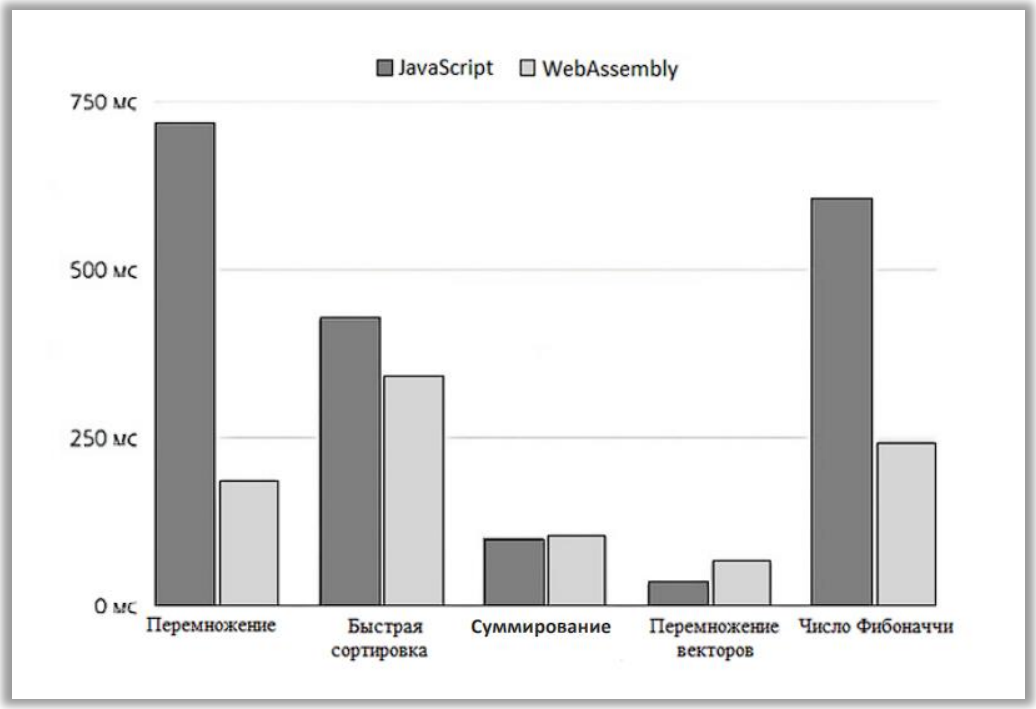

Рисунок 3 - Сравнение скорости выполнения функций

А.И. Рудаков в результатах научно-практической конференции сообщает, что [10]:

Технология WebAssembly пока активно развивается, и она не слишком активно применяется в пользовательских приложениях. Но она дает достаточно неплохой прирост к производительности, при правильном использовании, а в противном случае производительность может даже

ухудшиться. Также есть проблемы с производительностью на мобильных устройствах.

И. М. Струнин рассмотрел возможность представления грид-системы в виде веб-сервиса с вычислительными узлами на основе веб-браузеров, с применением WebAssembly для повышения производительности, реализовал пример такой архитектуры и выполнил замеры производительности его узлов [11]. В таблице 2 приведены результаты измерений.

Таблица 2 – Время выполнения тестов узлами (мс)

	C#	WebAssembly	JavaScript
Последовательно	16,46	18,57	29,81
Параллельно	5,40	6,19	10,23

К.П. Трубаев отмечает первоклассную совместимость средств языка программирования Rust с технологией WebAssembly [12].

Д.А. Филисов проводит исследование производительности приложения, использующий вычислительный сервис AWS Lambda [13]:

Разработка кода, который может эффективно использовать параллелизм, может представлять трудности, но награда существенна — высокоэффективное приложение, способное обрабатывать увеличенный трафик. Хотя писать код в вычислительной среде, подобной AWS Lambda, относительно просто, где каждый запрос может быть изолирован от других, недостаток заключается в невозможности эффективно использовать несколько параллельных запросов в рамках одной системы параллелизма. Однако выбор в пользу бессерверных вычислительных решений, таких как AWS App Runner или AWS Fargate, предоставляет возможность параллелизма на стороне приложения, позволяя внедрять эффективные шаблоны, которые полностью используют параллелизм.

Д. Шабан провел анализ производительности WebAssembly по сравнению с JavaScript и привел примеры успешных приложений, реализованных с помощью WebAssembly такое как Google Earth [14].

А.А. Шатоба в своей статье рассмотрел проблему повышения эффективности программного кода на языке программирования JavaScript, а именно времени выполнения. Автор реализовал пример системы оптимизации исходного кода в целях ускорения его выполнения [15].

2 АНАЛИЗ ЗАРУБЕЖНЫХ НАУЧНЫХ ИСТОЧНИКОВ ПО ТЕМЕ ИССЛЕДОВАНИЯ

В своей статье N.T. Asegehegn рассмотрел текущее состояние технологии, при котором выполняется сборка исходного кода Rust в бинарный модуль WebAssembly. Автор делает вывод, что [16]:

Текущий этап развития WebAssembly недостаточен для полноценной замены JavaScript при разработке PWA. Существуют ограничения, которые необходимо устранить, такие как отсутствие сборщика мусора и невозможность прямого доступа к DOM и Web-API. Однако Wasm вполне может стать стандартом при создании веб-приложений. Даже при всех ограничениях Wasm реализованный пример работал почти так же хорошо, как версия приложения на ReactJS. Чрезмерное использование памяти может стать существенным ограничением, что особенно актуально при ориентации на мобильные устройства. Скорее всего, эта ситуация изменится по мере того, как будет внедряться больше функций для WebAssembly.

Авторы J. De Macedo, R. Abreu, R. Pereira, J. Saraiva, исследуют производительность исполняемого кода между JavaScript и WebAssembly [17]. Ими были изучены приложения — эмулятор консоли «Gameboy», WasmBoy и средство просмотра и редактирования документов PDF, PSPDFKit, а также велось наблюдение за потреблением энергии и временем выполнения десяти микротестов с тремя разными наборами входных данных в трех популярных браузерах — Google Chrome, Mozilla Firefox, и Microsoft Edge. Результаты показали, что производительность Wasm варьируется в зависимости от того, рассматриваем ли мы реальные тесты или микротесты. В некоторых случаях, JS может быть эффективнее и быстрее, чем Wasm на микротестах, однако в реальных приложениях производительность Wasm намного выше. На микротестах Wasm показал большую производительность в Google Chrome и Microsoft Edge, в то время как в Mozilla Firefox, JS показал лучшие результаты

с большой разницей в большинстве случаев. Тем не менее, для реальных приложений, Wasm продемонстрировал значительное превосходство над JS [17].

В своей статье A. Hilbig, D. Lehmann, M. Pradel, выполнили анализ используемых языков для формирования бинарных модулей WebAssembly безопасность этих бинарных модулей [18]. Авторы использовали пакеты исходных кодов из открытых источников, таких как GitHub и инструментов, которые предоставляют менеджеры пакетов. На рисунке 4 представлено соотношение языков программирования, которые используют WebAssembly.

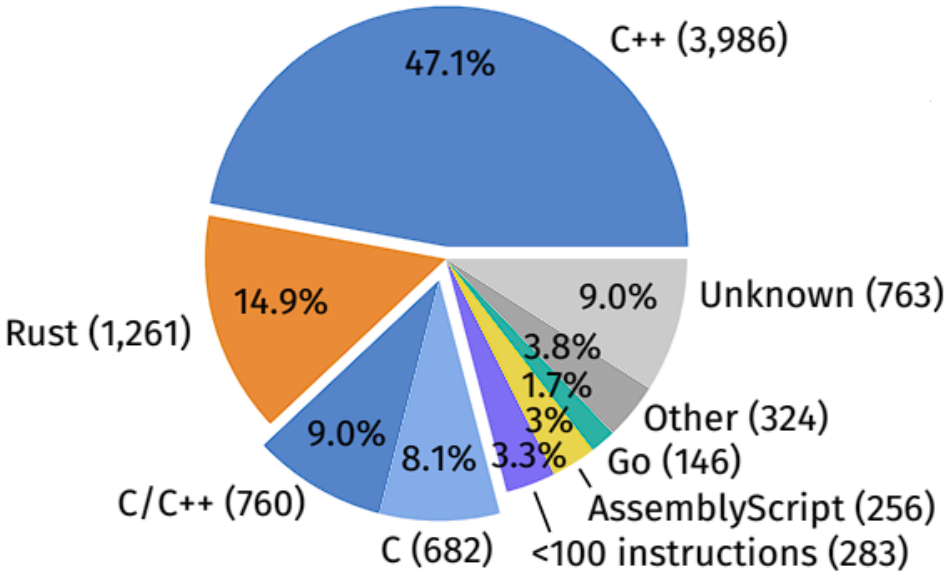

Рисунок 4 - Соотношение языков программирование, которые используются для WebAssembly

В этом исследовании авторы A. Jangda, B. Powers, E. D. Berger, A. Guha, представили полный разбор показателей эффективности WebAssembly. Авторы разработали BROWSIX-WASM, обновленную версию BROWSIX, а также BROWSIX-SPEC, набор инструментов, предназначенный для подробного исследования производительности, который позволяет им проводить тесты SPEC CPU2006 и CPU2017 как WebAssembly-приложения в

Chrome и Firefox. Оказалось, что средняя задержка WebAssembly по отношению к нативному коду в тестах SPEC в 1,55 раза больше для Chrome и в 1,45 раз больше для Firefox, в то время как максимальные задержки составляют в 2,5 раза больше в Chrome и в 2,08 раз больше в Firefox. Они анализируют причины этих различий в производительности и предлагают практические советы для будущих оптимизационных проектов [19].

На научно-практической конференции K. I. D. Kyriakou, N. D. Tselikas, рассмотрели тенденции и методы разработки приложений Node.js, от этапа прототипирования до момента, когда требуется оптимизация и улучшение производительности. Также были рассмотрены потенциальные преимущества использования языка программирования Rust по сравнению с более традиционными C/C++. Обсуждались сходства в рабочих процессах JS и Rust, и возможности для обеспечения совместимости. Исследование завершилось сравнительной оценкой производительности каждого подхода при различных нагрузках. В случаях, когда требуются сложные вычисления, Rust показал значительное улучшение в эффективности использования системных ресурсов Node.js при минимальных усилиях. Также было отмечено, что с Rust легко достигается взаимодействие между модулями, и они свободны от проблем с памятью и сбоями данных, которые были характерны для кода на C/C++ в прошлом. Авторы исследования обнаружили, что переход на Rust довольно прост, объясняя это отсутствием классов и наличием функционального программирования. По мнению участников, Rust больше удовлетворяет потребности разработчиков на JS, чем C/ C++ [20].

В статье F. Oliveira, J. Mattos, рассматривают, может ли технология WebAssembly повысить эффективность выполнения JavaScript в контексте интернета вещей. Для этого авторы провели ряд тестов, направленных на измерение потребления ресурсов (процессорное время, объем используемой памяти, расход заряда аккумулятора). Результаты тестов показывают, что использование WebAssembly значительно повышает производительность

JavaScript по всем рассмотренным параметрам, особенно в отношении экономии заряда аккумулятора. В устройствах с автономным питанием и ограничениями по ресурсам, расход энергии может стать критическим фактором, а не только скорость выполнения кода. WebAssembly представляет собой многообещающую технологию, которую можно адаптировать для различных контекстов, в том числе и для среды Интернета вещей. Благодаря скомпилированному коду и оптимизации потребления ресурсов, WebAssembly способен обеспечить необходимый компромисс между производительностью и расходом энергии. В будущих исследованиях авторы планируют рассмотреть возможность использования WebAssembly другими интерпретаторами JavaScript. Кроме того, авторы намереваемся углубить анализ использования памяти в данной технологии, так как это является одним из ограничений WebAssembly и открывает возможности для изучения возможностей использования JavaScript в качестве основного языка для окружения интернета вещей. [21].

ЗАКЛЮЧЕНИЕ

В ходе выполнения научной исследовательской работы было проанализировано влияние использования бинарных модулей WebAssembly приложениями JavaScript на примерах вычислительных задач с различными языками программирования.

Анализ источников показал, технология WebAssembly не может быть полной заменой JavaScript, однако при правильном выборе компилируемого языка для указанной задачи может дать прирост производительности всего приложения.

В ходе практики индивидуальное задание было полностью выполнено.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Бородин О. В., Егунов В. А., Плотников В. П. Особенности использования низкоуровневого процессорного кода с использованием Webassembly // Прикаспийский журнал: управление и высокие технологии. 2022. № 2. С. 68-83. URL: https://cyberleninka.ru/article/n/osobennosti-ispolzovaniya-nizkourovnevogo-protsessornogo-koda-s-ispolzovaniem-webassembly

2. Григоренко А. В. Некоторые вопросы использования технологии webassembly и web usb в современных браузерах // Аллея науки. 2019. Т. 2. № 2. С. 903-908. URL: https://alley-science.ru/domains_data/files/Collection_of_journals/Fevral%202%20tom.pdf#page=903

3. Гридин В. Н., Анисимов В. И., Васильев С. А. Методы повышения производительности современных Веб-приложений // Известия Южного федерального университета. Технические науки. 2020. № 2. С. 193-200. URL: https://cyberleninka.ru/article/n/metody-povysheniya-proizvoditelnosti-sovremennyh-veb-prilozheniy

4. Еремин Д. О., Холмогорова Е. И. Особенности применения Blazor WebAssembly для разработки веб-приложений // Инновационные технологии в технике и образовании. 2021. С. 320-325. URL: https://disk.yandex.ru/i/3gJqJ7ykIgk77w

5. Кособродов С. К. Сравнительный анализ производительности assemblyscript и javascript в продвинутых вычислительных алгоритмах // Инновационная наука. 2023. № 12-1. С. 29-47. URL: https://disk.yandex.ru/i/z6yzrJX_2gqFew

6. Лымарев А. А. Использование технологии React Native и WebAssembly для разработки мобильных приложений // Информационные технологии и автоматизация управления. 2019. С. 181-184. URL: https://disk.yandex.ru/i/cFMHYJlMugUr0g

7. Любицкий Н. И., Мальцев К. Р. Сравнение работы алгоритма построения диаграмм вороного с помощью триангуляции делоне на языках программирования javascript, c++, rustи go // Программная инженерия: современные тенденции развития и применения (пи-2020). 2020. С. 47-50. URL: https://disk.yandex.ru/i/0VElpyniVV-ibQ

8. Овчаров З. А. Методы оптимизации браузерных вычислений на основе распараллеливания потоков // Вестник технологического университета. 2023. Т. 26. № 11. С. 205-209. URL: https://disk.yandex.ru/i/v4J_VAPmxPPzxw

9. Рокотянская В. В., Абрамов В. С. Исследование WebAssembly и сравнение производительности с JavaScript // Вестник Астраханского государственного технического университета. Серия: Управление, вычислительная техника и информатика. 2023. № 2. С. 93-100. URL: https://cyberleninka.ru/article/n/issledovanie-webassembly-i-sravnenie-proizvoditelnosti-s-javascript

10. Рудаков А. И. WebAssembly // Молодежь и современные информационные технологии: сборник трудов XVII Международной научно-практической конференции студентов, аспирантов и молодых учёных (Томск, 17-20 февраля 2020 года). 2020. С. 368-369. URL: https://earchive.tpu.ru/bitstream/11683/62127/1/conference_tpu-2020-C04_p368-369.pdf

11. Струнин И. М. Один метод повышения производительности вычислений в веб-браузере // Системы компьютерной математики и их приложения. 2020. № 21. С. 221-225. URL: https://disk.yandex.ru/i/6Er1_7w3AW4K_w

12. Трубаев К. П. Современные языки программирования на примере языка rust // Образование. Наука. Производство. 2022. С. 197-200. URL: https://disk.yandex.ru/i/VAqoimAtQJ57Sg

13. Филисов Д. А. Высвобождение производительности: глубокое погружение в приложения с высокой нагрузкой // Международный научный журнал

инновационная наука. 2023. С. 37-47. URL: https://aeterna-ufa.ru/sbornik/IN-2023-10-2.pdf#page=37

14. Шабан Д. Webassembly как успешный пример реализации концепции кроссплатформенности // Научно-технические инновации и веб-технологии. 2022. № 1. С. 32-38. URL: https://disk.yandex.ru/i/O7jA0WQ0AtpU3w

15. Шатоба А. А. Средства повышения производительности javascript-кода // Проспект свободный-2020. 2020. С. 77-79. URL: https://disk.yandex.ru/i/aplaz3Lxjt7A-w

16. Asegehegn, N. T. Evaluation of Rust and WebAssembly when building a Progressive Web Application: An analysis of performance and memory usage: An analysis of performance and memory usage. 2022. 82 p. URL: https://www.diva-portal.org/smash/get/diva2:1668630/FULLTEXT01.pdf

17. De Macedo, J., Abreu, R., Pereira, R., Saraiva, J. WebAssembly versus JavaScript: Energy and Runtime Performance // 2022 International Conference on ICT for Sustainability (ICT4S). 2022. P. 24-34. URL: http://repositorio.inesctec.pt/bitstreams/0870fb76-d463-456b-9e34-5b33bb7c0dd1/download

18. Hilbig, A., Lehmann, D., Pradel, M. An empirical study of real-world webassembly binaries: Security, languages, use cases // Proceedings of the web conference 2021. 2021. P. 2696-2708. URL: https://www.software-lab.org/publications/www2021.pdf

19. Jangda, A., Powers, B., Berger, E. D., Guha, A. Not so fast: Analyzing the performance of {WebAssembly} vs. native code // 2019 USENIX Annual Technical Conference (USENIX ATC 19). 2019. P. 107-120. URL: https://www.usenix.org/system/files/atc19-jangda.pdf

20. Kyriakou, K. I. D., Tselikas, N. D. Complementing JavaScript in High-Performance Node. js and Web Applications with Rust and WebAssembly // Electronics. 2022. Vol. 11. N. 19. P. 3217. URL: https://www.mdpi.com/2079-9292/11/19/3217/pdf

21. Oliveira, F., Mattos, J. Analysis of WebAssembly as a strategy to improve JavaScript performance on IoT environments // Anais Estendidos do X Simpósio Brasileiro de Engenharia de Sistemas Computacionais. 2020. P. 133-138. URL: https://sol.sbc.org.br/index.php/sbesc_estendido/article/download/13102/12955/