

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО АВТОНОМНОГО ОБРАЗОВАТЕЛЬНОГО УЧРЕЖДЕНИЯ 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: Экспериментальное исследование производительности модулей
WebAssembly, реализованных на языке программирования Go.

Обучающийся Валиуллин А.Р., Р4107

Согласовано:

Руководитель практики от университета:

Государев И. Б., доцент,

кандидат педагогических наук, ФПИиКТ

Практика пройдена с оценкой отлично

Дата 29.06.2024

Санкт-Петербург

2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 АНАЛИЗ ИСТОЧНИКОВ ПО ТЕМЕ ИССЛЕДОВАНИЯ.....	5
2 ИНИЦИАЛИЗАЦИЯ И ИСПОЛНЕНИЕ WEBASSEMBLY В БРАУЗЕРЕ	7
3 СРАВНИТЕЛЬНАЯ ОЦЕНКА ПРОИЗВОДИТЕЛЬНОСТИ МОДУЛЕЙ WEBASSEMBLY И JAVASCRIPT	11
ЗАКЛЮЧЕНИЕ	15
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	16

ВВЕДЕНИЕ

В современном мире веб-приложения становятся все более сложными и требовательными к производительности. Для оптимизации работы веб-приложений используют различные технологии, одной из которых является WebAssembly - стандарт веб-технологий, позволяющий выполнять код на других языках программирования в браузере с производительностью, сравнимой с нативным исполнением.

Язык программирования Go также известен своей эффективностью и производительностью. Исследование производительности модулей WebAssembly, реализованных на языке программирования Go, имеет большое практическое значение для оптимизации веб-приложений. В данной работе будет проведено экспериментальное исследование с целью оценки производительности таких модулей и сравнения результатов с JavaScript.

Исследование направлено на выявление особенностей работы модулей WebAssembly, созданных на языке Go, и их влияния на производительность веб-приложений.

Актуальность работы обусловлена растущим интересом к технологии WebAssembly, которая предоставляет возможность запуска высокопроизводительного кода веб-приложений на различных платформах. Разработка веб-приложений с использованием WebAssembly и языка программирования Go представляет собой перспективное направление, которое требует проведения исследований по оптимизации производительности модулей WebAssembly.

Целью НИР является проведение экспериментального исследования производительности модулей WebAssembly, реализованных на языке программирования Go, с целью определения их эффективности и возможностей применения в различных веб-приложениях.

Объектом исследования являются модули WebAssembly, реализованные на языке программирования Go.

Предметом исследования является производительность модулей WebAssembly, написанных на языке программирования Go, в сравнении с реализациями на JavaScript.

Задачи исследования:

- 1) анализ существующих подходов к созданию модулей WebAssembly на языке программирования Go,
- 2) разработка тестового окружения для проведения экспериментальных исследований,
- 3) реализация модулей WebAssembly на языке программирования Go,
- 4) проведение экспериментов и сравнение производительности модулей WebAssembly, реализованных на Go, с библиотечными модулями JavaScript,
- 5) анализ результатов исследования и выявление перспектив дальнейшего развития данного подхода к веб-разработке.

1 АНАЛИЗ ИСТОЧНИКОВ ПО ТЕМЕ ИССЛЕДОВАНИЯ

В статьях «Функциональный подход к измерению вклада программируемых решений в производительность программ» и «WebAssembly versus JavaScript» приводится набор эталонных задач, которые используют для анализа производительности программных модулей с использованием различных языков программирования [1, 2]. В таблице 1 приведены время их выполнения для языков программирования Go, JavaScript.

Таблица 1 – Время выполнения задач (сек)

Задача	JavaScript	Go
fannkuch-redux	11,63	11,77
n-body	8,62	7,00
spectral-norm	5,38	5,33
mandelbrot	63,05	5,01
pidigits	12,45	1,34
regex-redux	5,57	25,87
fasta	39,46	3,74
k-nucleotide	38,11	7,53
reverse-complement	15,95	2,24
binary-trees	9,00	27,68

Результаты из таблицы 1 представлены на рисунке 1. Всего лишь в 3 из 10 вычислительных задач производительность языка программирования Go уступает задачам, реализованным на языке программирования JavaScript.

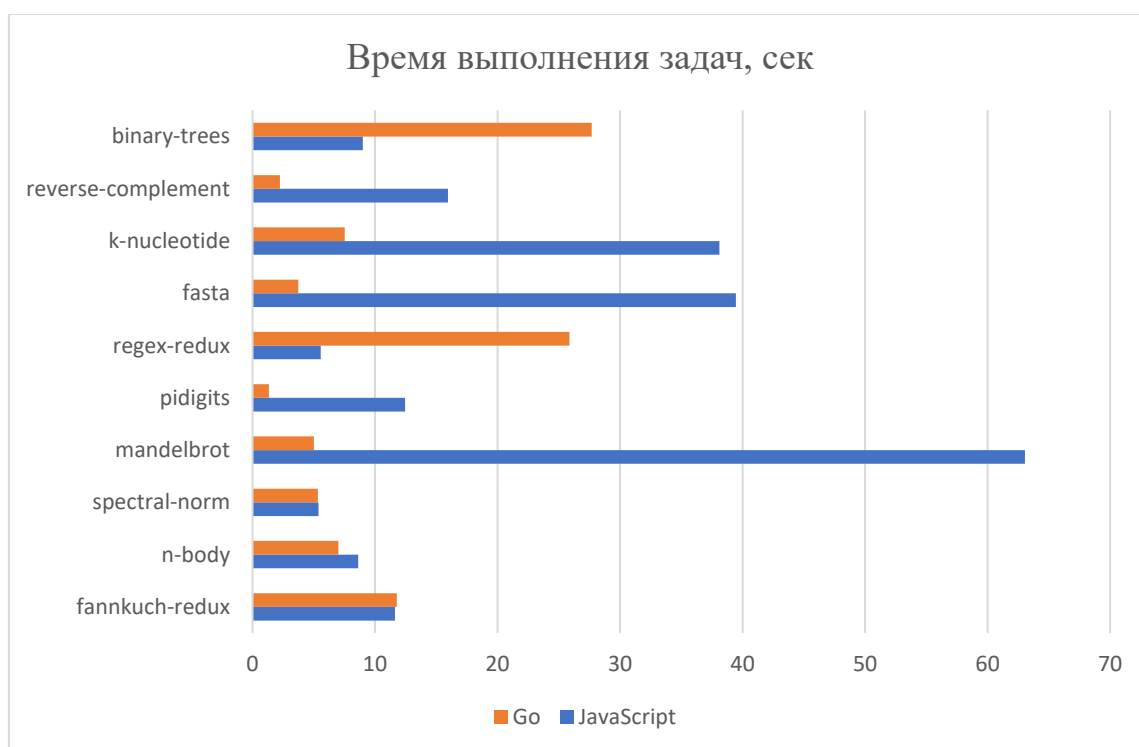


Рисунок 1 - Время выполнения задач

Несмотря на разнообразие и сложность задач, их выполнение не позволяет однозначно определить скорость работы кода. Это связано с тем, что реализация кода в задачах может быть разной. По этой причине авторы используют наиболее простые в реализации задачи, такие как: перемножение, быстрая сортировка, суммирование, перемножение векторов, число Фибоначчи [3, 4, 5, 6, 7].

Используя, источники определим наиболее удобные для равнозначной реализации вычислительные задачи и определим необходимые функции:

- рекурсивная функция Фибоначчи,
- функция перемножение целых чисел,
- функция перемножение векторов,
- функция факторизации числа.

2 ИНИЦИАЛИЗАЦИЯ И ИСПОЛНЕНИЕ WEBASSEMBLY В БРАУЗЕРЕ

Сначала браузер загружает HTML-страницу, на которой выполняется JavaScript. Затем JavaScript выполняет загрузку WebAssembly-модуля. После загрузки модуля создается экземпляр модуля, через который можно вызывать экспортируемые функции [8]. Общая схема выполнения WASM-модуля представлена на рисунке 2.

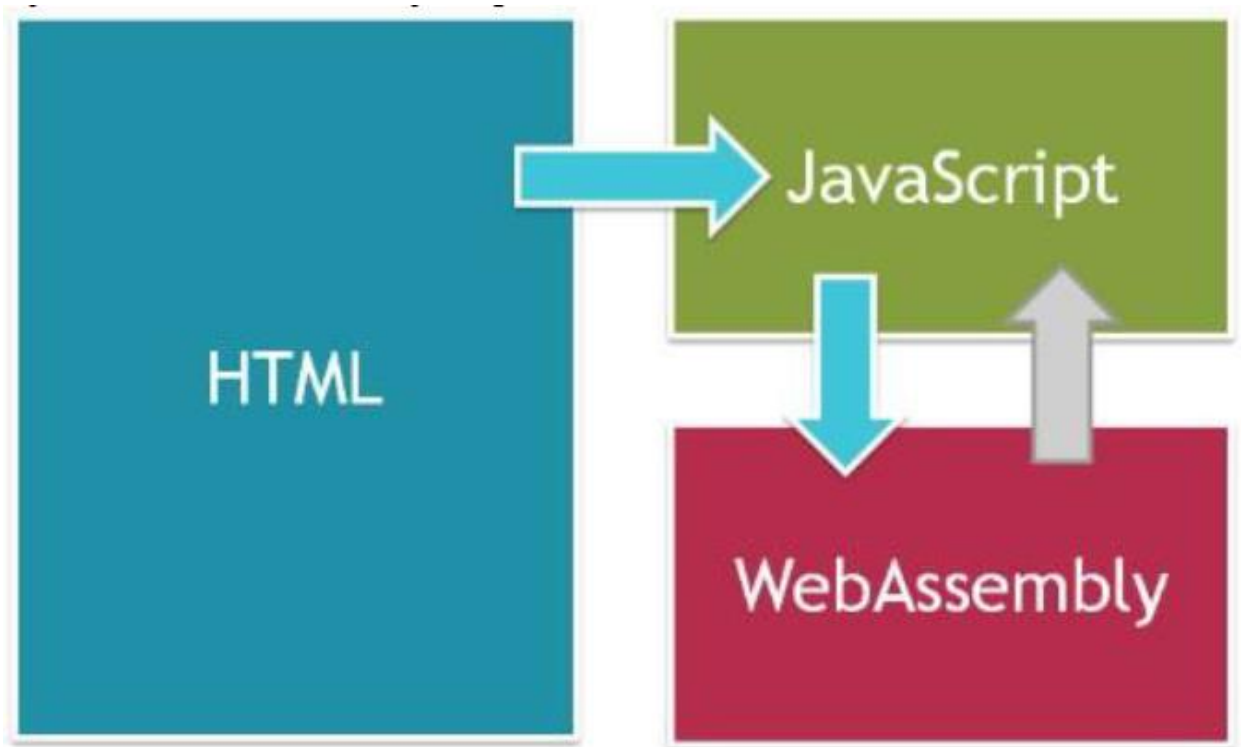
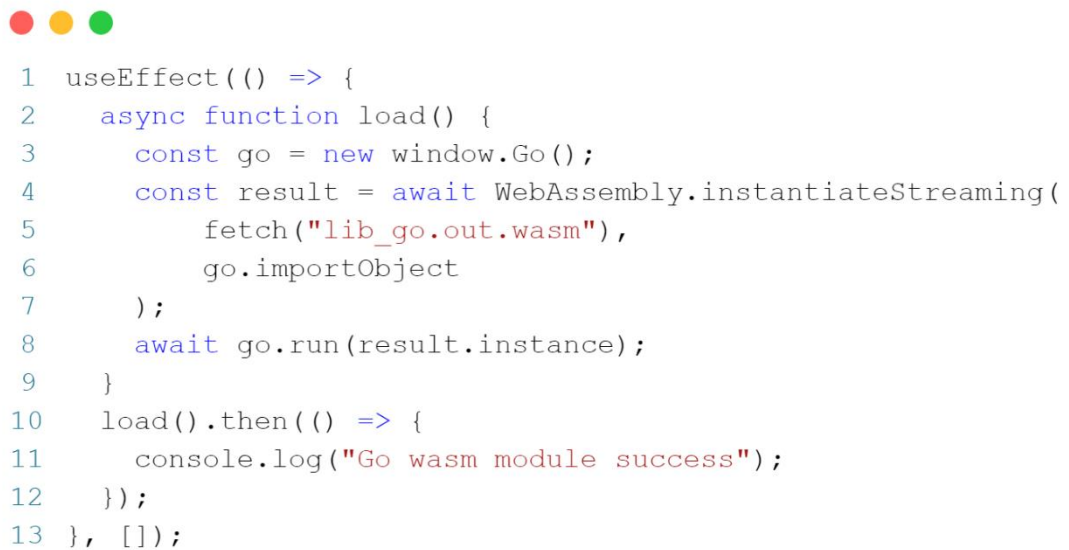


Рисунок 2 - Общая схема выполнения WebAssembly в браузере

На рисунке 3 показан способ инициализации модуля WebAssembly. Для этого используется хук `useEffect` и асинхронная функция `load()`. Внутри функции `load()` создается экземпляр `Go`, загружается WebAssembly-модуль из файла «`lib_go.out.wasm`» с помощью функций `fetch` и `WebAssembly.instantiateStreaming`. Затем модуль запускается с помощью метода `go.run()`. Пустой массив вторым аргументом функции `useEffect` (`[]`) означает, что эффект должен быть выполнен только один раз после монтирования компонента. После выполнения функции `load()` вызывается метод `then()`, который выводит в консоль сообщение об успешной загрузке модуля WebAssembly.



```
1  useEffect(() => {
2    async function load() {
3      const go = new window.Go();
4      const result = await WebAssembly.instantiateStreaming(
5        fetch("lib_go.out.wasm"),
6        go.importObject
7      );
8      await go.run(result.instance);
9    }
10   load().then(() => {
11     console.log("Go wasm module success");
12   });
13 }, []);
```

Рисунок 3 - Способ инициализации модуля WebAssembly


Функции модуля WebAssembly совместно с аналогичными реализациями на JavaScript будут использованы для измерения времени их выполнения.

Замеры производительности исполняемого кода могут быть выполнены посредством интерфейса Performance. Этот интерфейс представляется движком браузера. Метод performance.now() возвращает временную метку в миллисекундах [9]. Для измерения времени выполнения кода, метод необходимо применить до и после тестируемого фрагмента и вычесть разницу из полученных временных меток (рисунок 4).



Рисунок 4 - Пример измерения скорости участка кода

На рисунке 5 представлен код инициализации функции обработчика. Данный код на Go представляет собой функцию `main()`, которая используется для создания WebAssembly-кода из Go. В данной функции происходит следующее: с помощью метода `js.Global().Set()` каждой из функций `fibonacciRecursive`, `fibonacciIterative`, `multiply`, `multiplyVector` и `factorize` присваивается соответствующая функция-обработчик, созданная на Go. Таким образом, данный код инициализирует функции в глобальном объекте JavaScript, которые будут обращаться к соответствующим функциям, написанным на Go, после компиляции в WebAssembly.



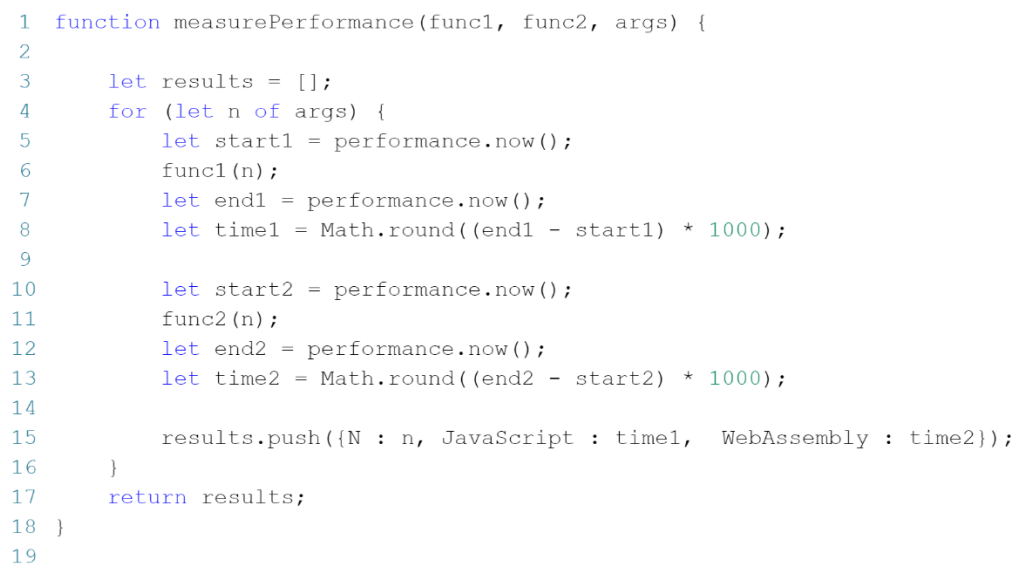
```
1 func main() {  
2     fmt.Println("Creating WebAssembly code from Go!")  
3     js.Global().Set("fibonacciRecursive", js.FuncOf(fibonacciRecursive))  
4     js.Global().Set("fibonacciIterative", js.FuncOf(fibonacciIterative))  
5     js.Global().Set("multiply", js.FuncOf(multiply))  
6     js.Global().Set("multiplyVector", js.FuncOf(multiplyVector))  
7     js.Global().Set("factorize", js.FuncOf(factorize))  
8     select {}  
9 }
```

Рисунок 5 – инициализации функции-обработчика

3 СРАВНИТЕЛЬНАЯ ОЦЕНКА ПРОИЗВОДИТЕЛЬНОСТИ МОДУЛЕЙ WEBASSEMBLY И JAVASCRIPT

Для проведения экспериментов по вычисления времени выполнения функций WebAssembly и JavaScript реализовано приложение с использованием библиотеки React [10].

На рисунке 6 представлена функция, которая формирует массив из времени выполняемых функций в эксперименте.



```
1 function measurePerformance(func1, func2, args) {
2
3   let results = [];
4   for (let n of args) {
5     let start1 = performance.now();
6     func1(n);
7     let end1 = performance.now();
8     let time1 = Math.round((end1 - start1) * 1000);
9
10    let start2 = performance.now();
11    func2(n);
12    let end2 = performance.now();
13    let time2 = Math.round((end2 - start2) * 1000);
14
15    results.push({N : n, JavaScript : time1, WebAssembly : time2});
16  }
17  return results;
18 }
19
```

Рисунок 6 – формирование данных эксперимента

На рисунках 7, 8 представлен пользовательский интерфейс приложения, используемый для проведения экспериментов. Выполнение эксперимента происходит путем нажатия на кнопку «Начать эксперимент». Сохранение результатов эксперимента выполняется путем нажатия на кнопку «Сохранить в CSV».

Исходный код приложения опубликовано на сервисе GitHub по ссылке: <https://github.com/arvaliullin/pyro.git>

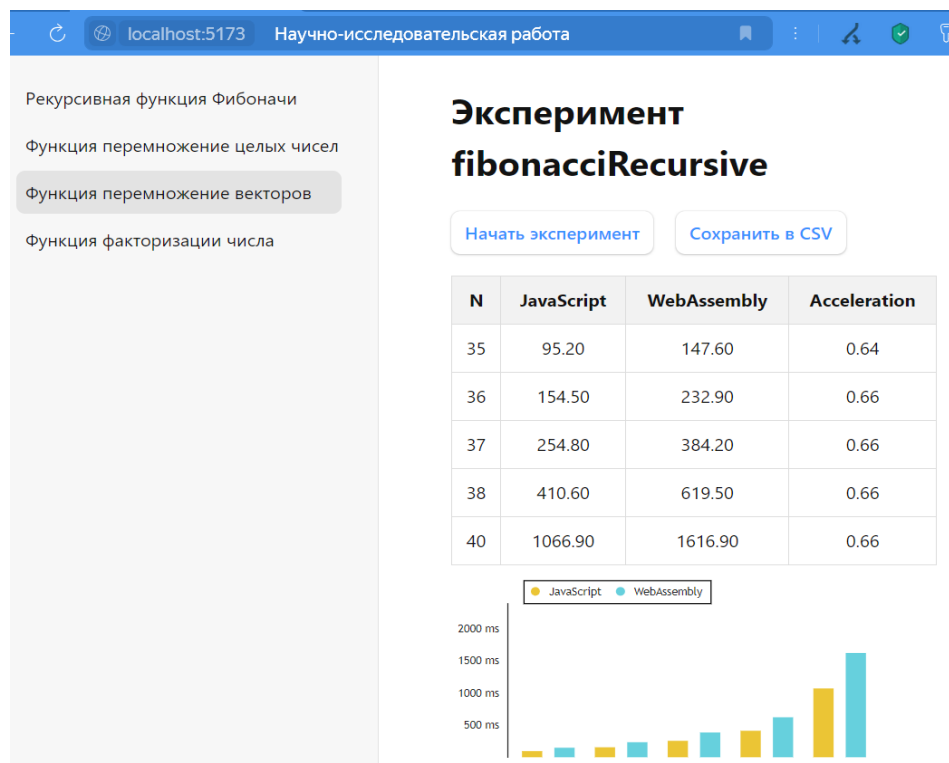


Рисунок 7 – формирование данных эксперимента

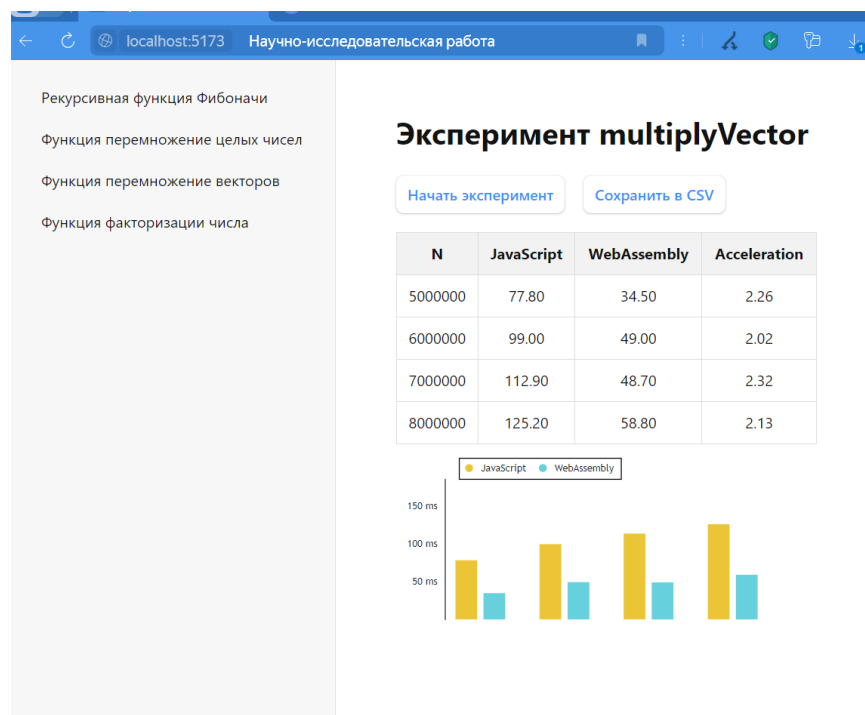


Рисунок 8 – формирование данных эксперимента

Из проведенных экспериментов в таблицах 2, 3, 4, 5 по замеру времени выполнения функций для WebAssembly и JavaScript можно сделать следующие практические выводы.

Ускорение выполнения функций при использовании WebAssembly по сравнению с JavaScript наблюдается не во всех измеренных случаях.

В случае функции факторизации числа ускорение выполнения при использовании WebAssembly составляет от 1,45 до 1,59, что означает приблизительно 1,5-кратное увеличение скорости выполнения по сравнению с JavaScript.

Для функции перемножение векторов ускорение составляет от 2,18 до 2,89, что является заметным улучшением производительности при использовании WebAssembly.

В случае функции перемножение целых чисел наблюдается ускорение от 0,29 до 0,50.

Для рекурсивной функции Фибоначчи ускорение составляет от 0,61 до 0,65, что также указывает на условную эффективность WebAssembly в данном случае.

Таблица 2 – Время выполнения факторизации числа N (мс)

N	JavaScript	WebAssembly	Ускорение
184382976303	74.00	51.00	1.45
210987654321	1382.50	866.70	1.60
123456789101	82.80	52.30	1.58
987654321009	2145.70	1357.70	1.58

Таблица 3 – Время выполнения перемножения векторов (мс)

N	JavaScript	WebAssembly	Ускорение
5000000	86.70	36.10	2.40
6000000	94.70	43.50	2.18
7000000	114.70	46.90	2.45
8000000	159.10	55.10	2.89

Таблица 4 – Время выполнения перемножения чисел (мс)

N	JavaScript	WebAssembly	Ускорение
50000000	25.00	86.40	0.29

600000000	28.70	57.80	0.50
700000000	33.30	69.20	0.48
800000000	38.90	78.00	0.50

Таблица 5 – Время выполнения вычисления числа Фибоначчи (мс)

N	JavaScript	WebAssembly	Ускорение
35	94.60	154.90	0.61
36	153.70	245.10	0.63
37	248.60	389.00	0.64
38	403.50	621.90	0.65

ЗАКЛЮЧЕНИЕ

При выполнении вычислительно сложных задач WebAssembly демонстрирует значительное улучшение производительности по сравнению с JavaScript, однако для определенных типов задач может потребоваться дополнительная оптимизация. При использовании языка программирования Go и его инструментария малая эффективность может быть связана с условно дорогим вызовом функции.

В ходе практики индивидуальное задание было полностью выполнено.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Андреева Т. А., Городняя Л. В. Функциональный подход к измерению вклада программируемых решений в производительность программ: препринт. Новосибирск: Сибирское отделение РАН Институт систем информатики им. А.П. Ершова, 2022. 62 с. URL: https://www.iis.nsk.su/files/preprint/preprint_187.pdf
2. De Macedo J., Abreu R., Pereira R. et al. 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>
3. 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 (58). С. 68-83. URL: <https://cyberleninka.ru/article/n/osobennosti-ispolzovaniya-nizkourovneвого-protseссornого-koda-s-ispolzovaniem-webassembly>
5. Ленкин А. В. Обзор использования Goroutines языка программирования Go в целях ускорения работ программного обеспечения // Постулат. 2019. № 9 сентябрь. С. 1-4. URL: <https://pgusa.tmweb.ru/index.php/Postulat/article/viewFile/2859/2903>
6. 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>
7. Jangda A., Powers B., Berger E. D. et al. 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://scholar.google.com/scholar_url?url=https://www.usenix.org/system/files/atc

19-jangda.pdf&hl=ru&sa=T&oi=gsb-
gga&ct=res&cd=0&d=15775910724547739499&ei=16flZay0DbrKsQLf1KKwAQ
&scisig=AFWwaea1RLEKQHZq8E5ADsyoh4-7

8. 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
9. Selakovic M., Pradel M. Performance issues and optimizations in javascript: an empirical study //Proceedings of the 38th International Conference on Software Engineering. – 2016. – С. 61-72.
10. React / [Электронный ресурс] // React : [сайт]. — URL: <https://react.dev/> (дата обращения: 17.06.2024).