# Исследование и сравнительный анализ производительности AssemblyScript

*А.С. Лабоскин*

*Acquire, San Francisco, USA*

**Аннотация:** В мире веб-разработки возрастает потребность в инструментах, способных обеспечивать высокую производительность клиентских приложений. В ответ на этот вызов была разработана технология WebAssembly, позволяющая компилировать различные языки программирования в бинарный формат, который затем исполняется в веб-браузерах. Новый язык программирования AssemblyScript предоставляет возможность создавать высокопроизводительные WebAssembly модули с использованием привычного для веб-разработчиков синтаксиса языка TypeScript. В данной работе проводится исследование WebAssembly и AssemblyScript, и сравнивается производительность AssemblyScript и JavaScript на примере четырех вычислительных алгоритмов. Результаты тестирования демонстрируют более высокую скорость выполнения AssemblyScript в большинстве задач, а также более стабильную производительность при выполнении в разных браузерах. Исследование подчеркивает актуальность применения AssemblyScript для оптимизации ресурсоемких операций при разработке веб-приложений.
**Ключевые слова:** assemblyscript, webassembly, wasm, javascript, front end, производительность, веб-приложения

## Введение

Язык программирования JavaScript, разработанный Бренданом Айком в 1995 году в качестве инструмента для добавления интерактивности веб-страницам, с течением времени превратился из простого скриптового языка в мощную платформу для разработки веб-приложений [1]. Универсальность и простота использования сделали JavaScript стандартом веб-разработки, и долгое время он оставался единственным языком программирования, поддерживаемым всеми основными веб-браузерами. С развитием стандарта ECMAScript и появлением таких технологий, как Node.js, область применения языка значительно расширилась. Это позволило разработчикам использовать его для создания серверных, десктопных и мобильных приложений.

В связи со значительным ростом мощностей вычислительных устройств Интернет перестал состоять исключительно из простых статичных веб-страниц. Значительная часть вычислений, ранее выполнявшихся на

сервере, мигрировала на сторону клиента. Современные веб-приложения имеют динамический контент и выполняют ресурсоемкие вычислительные операции в браузере. Однако, изначально JavaScript не разрабатывался для подобных решений, из-за чего он имеет ряд особенностей, негативно влияющих на его производительность.

Слабой стороной JavaScript является его динамическая типизация, которая позволяет определять и изменять тип переменной во время выполнения программы. Это обеспечивает гибкость и удобство разработки, но в то же время требует дополнительных проверок и преобразований в процессе исполнения, что приводит к снижению производительности по сравнению со статически типизированными языками, такими как C++, C# или Java.

Помимо этого, JavaScript является интерпретируемым языком программирования. Это означает, что его код выполняется построчно без предварительной компиляции. В отличии от компилируемых языков, где преобразование в машинный код происходит заранее, в интерпретируемом языке анализ и преобразование кода происходит в процессе выполнения, что негативно влияет на скорость его работы. Современные веб-браузеры используют JIT-компилятор (Just-In-Time) для решения этой проблемы, однако этого недостаточно для достижения уровня производительности языков с предварительной компиляцией.

Наконец, несмотря на то что современные стандарты JavaScript предоставляют возможности для асинхронного программирования, такие как Promise и операторы async/await, сам язык по-прежнему остается однопоточным. Это означает, что выполнение ресурсоемких операций может блокировать поток, что в свою очередь блокирует интерфейс страницы и негативно сказывается на пользовательском опыте.

В ответ на вышеуказанные ограничения и необходимость повышения производительности веб-приложений в индустрии начался поиск альтернативных решений. Одной из наиболее перспективных технологий в этом направлении стал WebAssembly — бинарный формат инструкций, спроектированный как портативная цель компиляции других языков программирования, который затем исполняется в браузере с производительностью, близкой к нативной. Одновременно с этим был представлен новый язык программирования AssemblyScript, который использует привычный для JavaScript разработчиков синтаксис, но при этом компилируется в высокопроизводительный код WebAssembly.

Целью данной работы является исследование технологии WebAssembly и языка программирования AssemblyScript, сравнительный анализ его производительности по отношению к JavaScript и определение областей его применения при разработке веб-приложений.

## WebAssembly

WebAssembly (WASM) — это бинарный формат инструкций для стековых виртуальных машин, предназначенный для выполнения в веб-браузере. Он был разработан в качестве цели компиляции для других языков программирования, таких как C, C++ и Rust, и позволяет выполнять их код в веб-клиенте со сравнимой с нативными приложениями производительностью [2].

Разработка WebAssembly началась в 2015 году в рамках World Wide Web Consortium (W3C). Над проектом работали специалисты из компаний, разрабатывающих ведущие веб-браузеры, включая Google, Apple, Mozilla и Microsoft. В 2017 году было объявлено о завершении разработки, и появились первые версии браузеров с поддержкой WASM. Однако развитие технологии продолжается, и с каждым годом ее возможности расширяются. По состоянию на ноябрь 2023 года уровень поддержки WebAssembly

составляет 97,28% среди всех установленных браузеров [3]. Список версий основных веб-браузеров, поддерживающих WASM модули представлен в таблице №1.

Таблица № 1

Браузеры с поддержкой WebAssembly

| | Chrome | Safari | Firefox | Edge | Opera |
|---|---|---|---|---|---|
| Версии | 57+ | 11+ | 52+ | 16+ | 44+ |
| Дата выхода | 9.03.2017 | 19.09.2017 | 7.03.2017 | 17.10.2017 | 21.03.2017 |

Код WebAssembly определяет абстрактное синтаксическое дерево (AST) и содержит в себе инструкции, структуры данных, функции и другие компоненты, необходимые для выполнения программы. В то время, как JavaScript требует анализа и интерпретации перед выполнением, WASM модули быстро декодируются и компилируются в машинный код. Бинарный формат обеспечивает маленький размер файла и его быструю загрузку, а для облегчения разработки и отладки существует также текстовое представление WASM — WebAssembly Text Format (WAT). Оно позволяет разработчикам просматривать и редактировать модули в удобном для чтения виде. Пример компиляции C++ кода в WAT представлен на рис. 1

```
int square(int value) {
  return value * value;
}
```

```
(module
 (table 0 anyfunc)
 (memory $0 1)
 (export "memory" (memory $0))
 (export "_Z6squarei" (func $_Z6squarei))
 (func $_Z6squarei (; 0 ;) (param $0 i32) (result i32)
  (i32.mul
   (get_local $0)
   (get_local $0)
  )
 )
)
```

Рис. 1. – Компиляции C++ кода в текстовый формат WebAssembly

Безопасность технологии обеспечивается выполнением каждого WASM модуля в изолированной песочнице внутри браузера. Это создает слой безопасности от потенциально вредоносного кода и гарантирует, что код, выполняемый внутри контекста WebAssembly не может прямо взаимодействовать с ресурсами системы или выполнять вредоносные действия. Помимо этого, WASM предоставляет строгую проверку типов и контроль доступа к памяти, что обеспечивает дополнительную безопасность его выполнения [4].

Несмотря на все преимущества, WebAssembly не имеет широкого набора сторонних библиотек и возможностей по взаимодействию с API браузера, а также может уступать JavaScript в производительности при выполнении некоторых типов задач. Кроме этого, WASM имеет более высокий порог вхождения, требуя от веб-разработчиков освоения других языков и понимания концепций низкоуровнего программирования.

## AssemblyScript

AssemblyScript — это мультипарадигменный язык программирования, основанный на TypeScript и предназначенный для компиляции в бинарный формат WebAssembly. Ключевая особенность AssemblyScript, отличающая его от других компилируемых в WASM языков, заключается в его синтаксическом сходстве с TypeScript и JavaScript. Это делает язык доступным для разработчиков, знакомых с экосистемой веб-разработки, и предоставляет им удобный инструмент для создания высокопроизводительных веб-приложений.

В отличие от TypeScript, который является надстройкой над JavaScript с добавлением типизации и дополнительными возможностями для разработки, AssemblyScript разрабатывается с учетом требований WebAssembly, что накладывает определенные ограничения. В TypeScript типы данных могут быть унаследованы автоматически или проигнорированы, а AssemblyScript

использует статическую типизацию, требующую от разработчиков явного определения типа каждой переменной, и имеет более строгую систему типов [5]. Например, вместо используемого в TypeScript типа number для всех численных переменных, AssemblyScript разделяет их на i32 (32-битное целое число), u32 (32-битное положительное целое число), f64 (64-битное число с плавающей точкой) и другие. Кроме этого, в AssemblyScript отсутствует поддержка многих привычных для JavaScript возможностей, которые не могут быть эффективно представлены в WASM, включая замыкания, асинхронные операции с использованием Promise API, прототипное наследование и работу с DOM деревом.

Среда разработки AssemblyScript включает эталонный компилятор ASC, который преобразует код в формат WebAssembly и осуществляет ряд оптимизаций, таких как удаление неиспользуемого кода (tree-shaking) и встраивание функций (inlining). Помимо этого, AssemblyScript имеет стандартную библиотеку, которая предоставляет базовые операции и структуры данных, аналогичные стандартной библиотеке JavaScript, но адаптированные под требования WASM.

На протяжении нескольких лет с момента появления AssemblyScript активно развивается и поддерживается сообществом разработчиков, что приводит к постоянному расширению его возможностей и повышению производительности. По состоянию на ноябрь 2023 года на сервисе GitHub размещено более 23 тысяч проектов, использующих AssemblyScript [6].

Тем не менее, стоит отметить, что WebAssembly и AssemblyScript никогда не рассматривались в качестве полноценной замены для JavaScript. Вместо этого, они предназначены для совместного использования при разработке веб-приложений при помощи WebAssembly JavaScript API, позволяющего выполнять вызовы функций и передавать данные между WASM и JavaScript кодом [7].

## Методика тестирования производительности

Для проведения сравнения производительности AssemblyScript и JavaScript при выполнении ресурсоемких вычислений был использован набор из 4 вычислительных алгоритмов, имеющих версии на обоих языках программирования с минимальным количеством отличий, обусловленных особенностями каждого из языков [8]. Код алгоритмов был доработан с учетом актуальных на данный момент версий и стандартов разработки на AssemblyScript и JavaScript. Полный список использованных алгоритмов:

- LU-разложение: представление матрицы в виде произведения нижней (L) и верхней (U) треугольных матриц, используемый для решения систем линейных уравнений;
- Быстрое преобразование Фурье: алгоритм ускоренного вычисления дискретного преобразования Фурье, применяемый для анализа частотных компонентов дискретных сигналов;
- Поиск в ширину: один методов обхода графа, позволяющий найти кратчайшие пути из одной вершины невзвешенного графа до всех остальных вершин;
- Умножение разреженной матрицы на вектор (SpMV): алгоритм матричного умножения, оптимизированный для матриц с преимущественно нулевыми элементами.

Измерение времени, затрачиваемого на выполнение каждой функции, проводилось с использованием браузерного интерфейса Performance и его метода performance.now(), возвращающего точную временную метку в миллисекундах. Данный метод вызывался до вызова функции и после нее, а результатом теста являлась разница между двумя значениями. Чтобы избежать искажения данных фоновыми процессами операционной системы и оптимизациями браузеров, результат вычислялся как среднее арифметическое значение 100 запусков каждой функции в миллисекундах.

Для проведения замеров использовался ноутбук Apple Macbook Pro 16 с процессором Apple M1 Pro и 16 Гб ОЗУ, работающий на операционной системе macOS 14.0 Sonoma. Тестирование проводилось в 3 наиболее популярных веб-браузерах: Google Chrome (версия 118), Mozilla Firefox (версия 118) и Apple Safari (версия 17.0).

## Результаты тестирования

Итоги замеров производительности представлены в таблице №2 и на рис. 2

Таблица № 2

Результаты тестирования

| | Время выполнения, с | | | | | |
|---|---|---|---|---|---|---|
| | Chrome | | Firefox | | Safari | |
| | AS | JS | AS | JS | AS | JS |
| LU-разложение | 42.97 | 59.92 | 45.69 | 76.37 | 42.15 | 52.31 |
| Быстрое преобразование Фурье | 63.55 | 43.07 | 70.46 | 36.07 | 54.01 | 18.33 |
| Поиск в ширину | 29.02 | 40.96 | 33.64 | 147.65 | 29.22 | 59.14 |
| Умножение разреженной матрицы на вектор | 19.26 | 61.76 | 21.78 | 142.13 | 20.52 | 125.27 |

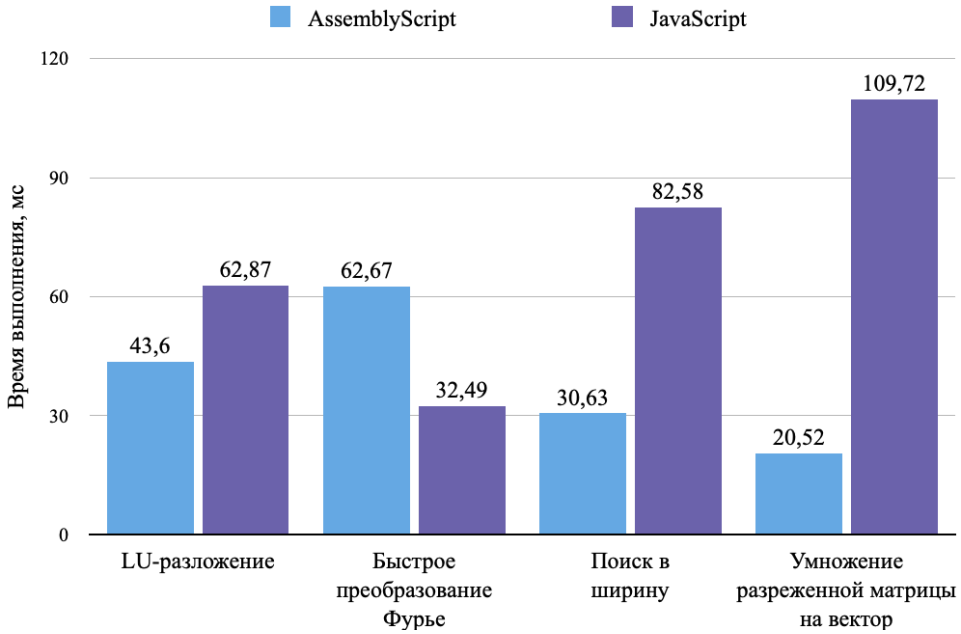


Рис. 2. – Результаты тестирования

По итогам тестирования было установлено, что AssemblyScript превосходит JavaScript по производительности в большинстве задач. При этом разница в скорости выполнения зависит от конкретного алгоритма и веб-браузера, достигая более чем шестикратного прироста в случае выполнения алгоритма умножения разреженной матрицы на вектор в браузере Apple Safari.

В то же время AssemblyScript выполнил Быстрое преобразования Фурье почти вдвое медленнее, чем JavaScript. С высокой долей вероятности это обусловлено тем, что AssemblyScript использует подсчет ссылок для управления памятью. В данном алгоритме происходит обход двумерного массива во вложенном цикле, что влечёт за собой создание и удаление ссылок на каждой итерации. Данная проблема, вероятно, будет устранена, когда в стандарт WebAssembly будет включён сборщик мусора (WasmGC), находящийся в разработке по состоянию на ноябрь 2023 года [9]. Однако этот результат говорит о существовании задач, для которых AssemblyScript не является предпочтительным решением.

Стоит отметить, что AssemblyScript показывает более стабильную производительность при выполнении в разных браузерах. В то время, как в AssemblyScript разница в скорости выполнения алгоритма не превышает 23%, в случае JavaScript этот показатель достигает 72%. Это обусловлено уникальным набором оптимизаций при выполнении JavaScript кода, используемых в каждом из веб-браузеров. WebAssembly использует AOT-компиляцию (Ahead-of-Time), что позволяет получать более предсказуемую и стабильную производительность с низкой зависимостью от среды исполнения.

Полученные в результате тестирования данные подтверждают гипотезу о более высокой производительности AssemblyScript относительно JavaScript при выполнении ресурсоемких вычислений. В связи с этим, применение AssemblyScript может быть целесообразным для оптимизации вычислительно интенсивных задач в таких областях, как работа с графикой, криптография, виртуальная реальность, 3D-рендеринг и видеоигры.

## Заключение

В результате проведенного исследования становится очевидным, что AssemblyScript представляет собой мощный инструмент для улучшения производительности веб-приложений в вычислительно интенсивных задачах. Высокая скорость выполнения ресурсоемких вычислений, продемонстрированная в ходе проведенного тестирования, обусловлена компиляцией языка в высокопроизводительный код WebAssembly. Синтаксическое сходство с TypeScript выделяет AssemblyScript среди других языков, компилируемых в WASM, и облегчает его интеграцию в существующие проекты, предоставляя разработчикам гибкость в выборе подходящих технологий для каждой конкретной задачи.

Однако, несмотря на его превосходство в производительности, AssemblyScript не предназначен для полной замены JavaScript в качестве

универсального решения для веб-разработки. Оптимальный подход заключается в применении разработанных на AssemblyScript WASM модулей для задач, требующих максимальной производительности. Для основной логики приложения и взаимодействия с DOM деревом при этом рекомендуется использовать JavaScript [10]. Такой симбиоз позволит объединить преимущества обоих языков программирования, обеспечивая высокую скорость работы AssemblyScript и широкие возможности экосистемы JavaScript.

Дальнейшие исследования в этом направлении могут включать расширение набора алгоритмов для тестирования, замеры потребления ресурсов процессора и памяти, а также сравнение производительности AssemblyScript с другими языками программирования, такими, как C++ и Rust, скомпилированными в формат WebAssembly.

## Литература

1. Балеев И.А., Земцов А.Н., Астахов Д.А. Методы обработки и визуализации данных в веб-интерфейсе с помощью библиотеки Dimensional Charting // Инженерный вестник Дона, 2017, №6. URL: ivdon.ru/uploads/article/pdf/IVD_2__6_baleev_zemtsov_astahov.pdf_7696387054.pdf

2. Haas A., Rossberg A., Schuff D. L., Titzer B. L., Holman M., Gohman D., Wagner L., Zakai A., Bastien JF. Bringing the Web up to Speed with WebAssembly // PLDI '23: ACM SIGPLAN Conference on Programming Language Design and Implementation, 2017. P. 185–200. URL: css.csail.mit.edu/6.858/2022/readings/wasm.pdf

3. "WebAssembly" | Can I use... Support tables for HTML5, CSS3, etc. URL: caniuse.com/?search=WebAssembly

4. Зернов, В. А. Анализ безопасности линейной памяти WebAssembly / В. А. Зернов // Вопросы устойчивого развития общества, 2022, № 4. URL: elibrary.ru/download/elibrary_48441770_74176309.pdf

5. Concepts | The AssemblyScript Book. URL: assemblyscript.org/concepts.html

6. Network Dependents - AssemblyScript/assemblyscript. URL: github.com/AssemblyScript/assemblyscript/network/dependents

7. Using the WebAssembly JavaScript API - WebAssembly | MDN. URL: developer.mozilla.org/en-US/docs/WebAssembly/Using_the_JavaScript_API

8. nischayv/as-benchmarks: AssemblyScript Benchmarks. URL: github.com/nischayv/as-benchmarks

9. Roadmap - WebAssembly. URL: webassembly.org/roadmap

10. Бетеев К.Ю., Муратова Г.В. Концепция Virtual Dom в библиотеке React.js // Инженерный вестник Дона, 2022, №3. URL: ivdon.ru/uploads/article/pdf/IVD_18__3_beteev_muratova.pdf_40562e973a.pdf

## References

1. Baleev I.A., Zemcov A.N., Astahov D.A. Inzhenernyj vestnik Dona, 2017, №6. URL: ivdon.ru/uploads/article/pdf/IVD_2__6_baleev_zemtsov_astahov.pdf_7696387054.pdf

2. Haas A., Rossberg A., Schuff D. L., Titzer B. L., Holman M., Gohman D., Wagner L., Zakai A., Bastien JF. PLDI '23: ACM SIGPLAN Conference on Programming Language Design and Implementation, 2017. pp. 185–200. URL: css.csail.mit.edu/6.858/2022/readings/wasm.pdf

3. "WebAssembly" | Can I use... Support tables for HTML5, CSS3, etc. URL: caniuse.com/?search=WebAssembly

4. Zernov, V. A. Voprosy ustojchivogo razvitija obshhestva, 2022, № 4. URL: elibrary.ru/download/elibrary_48441770_74176309.pdf

5. Concepts | The AssemblyScript Book. URL: assemblyscript.org/concepts.html

6. Network Dependents - AssemblyScript/assemblyscript. URL: github.com/AssemblyScript/assemblyscript/network/dependents

7. Using the WebAssembly JavaScript API - WebAssembly | MDN. URL: developer.mozilla.org/en-US/docs/WebAssembly/Using_the_JavaScript_API

8. nischayv/as-benchmarks: AssemblyScript Benchmarks. URL: github.com/nischayv/as-benchmarks

9. Roadmap - WebAssembly. URL: webassembly.org/roadmap

10. Beteev K.J., Muratova G.V. Inzhenernyj vestnik Dona, 2022, №3. URL:ivdon.ru/uploads/article/pdf/IVD_18__3_beteev_muratova.pdf_40562e973a.pdf