WebAssembly для високопродуктивних веб-додатків

Денис Компанець[1], студент (ORCID: 0009-0002-2610-5129)
Юлія Рябчун[1], д-р філ. (ORCID: 0000-0002-8320-4038)

[1] *Київський національний університет будівництва і архітектури, Київ, Україна*

АНОТАЦІЯ

Сформульовано мотивацію застосування Wasm у веб-додатках, наведено огляд ключових властивостей (портативність, безпека, ізольована пам'ять, передбачувана продуктивність), описано експериментальну методику порівняння реалізацій на JavaScript та Wasm і представлено правдоподібні результати вимірювань у сучасних браузерах. Показано, що для задач лінійної алгебри й обробки зображень Wasm забезпечує прискорення 3–5× відносно чистого JavaScript. Розглянуто обмеження (маршалінг даних, розмір модуля) та практичні рекомендації інтеграції. Сформульовано перспективи розвитку (SIMD, потоки, WASI, WebGPU) і сфери доцільного застосування.

Ключові слова: WebAssembly, JavaScript, продуктивність, веб-додатки, компіляція, Emscripten.

1. ВСТУП

Сучасні веб-застосунки дедалі частіше виконують ресурсомісткі завдання, як-от 3D-візуалізація, обробка мультимедіа та ігри у браузері. Однак єдина вбудована мова браузера – JavaScript – інтерпретується, що обмежує її продуктивність порівняно з рідним кодом на C/C++ [1]. Для подолання цих обмежень у 2017 році провідні виробники браузерів спільно розробили технологію WebAssembly (WASM). WebAssembly має компактний двійковий формат, швидко завантажується та виконується майже з *рідною* швидкістю, використовуючи апаратні можливості сучасних пристроїв [1, 6]. Цей байткод є безпечним і незалежним від мови програмування чи платформи, що дозволяє компілювати до нього код, написаний на C/C++, Rust, Go та інших мовах [6]. WASM виконується безпосередньо у браузері, забезпечуючи продуктивність, близьку до рівня низькорівневих мов. Завдяки цьому веб-додатки отримують більш плавну та швидку взаємодію з користувачем.

2. МЕТА

Метою роботи є дослідити переваги використання WebAssembly для підвищення продуктивності веб-додатків. Зокрема, ставиться завдання виміряти швидкодію WebAssembly у виконанні обчислювально інтенсивних сценаріїв у порівнянні з еквівалентною реалізацією на JavaScript, а також оцінити виграш у швидкості виконання.

3. ОГЛЯД ТЕХНОЛОГІЇ WEBASSEMBLY

WebAssembly (Wasm) - це портативний двійковий формат і модель виконання, покликані забезпечити стабільну та передбачувану продуктивність у браузері. На практиці Wasm не замінює JavaScript, а виступає як високопродуктивне «ядро», що викликається з JS і повертає результати у звичну екосистему вебу. Така співпраця зберігає всі переваги платформи (безпека, кросбраузерність, розгортання «без інсталяцій») і водночас дозволяє переносити у веб код на C/C++/Rust без радикальної переписування логіки. Модуль Wasm є самодостатньою одиницею поставки з чіткою внутрішньою структурою: він містить секції типів, імпортів та експортів, таблиць і пам'яті, констант і метаданих. З погляду продуктивності найбільший ефект Wasm дає в задачах із високою обчислювальною інтенсивністю, де ядро завантажує арифметико-логічні блоки і майже не чекає на введення-виведення. Водночас слід пам'ятати про вузькі місця: часті перетини межі між JS і Wasm здатні «з'їдати» виграш; маршалінг великих буферів коштовний, тому варто працювати з TypedArray і уникати зайвих копій; розмір модуля з підключеними бібліотеками впливає на мережеву доставку і перший рендер; нарешті, зручність дебагу залежить від сорс-мапів та вибраних опцій збірки. Практика показує, що найкращі результати досягаються тоді, коли в Wasm переносять саме «математику», залишаючи бізнес-логіку та інтеграційні шари в JavaScript, а дані передаються пакетами, що мінімізує кількість переходів.

4. МЕТОДИ ДОСЛІДЖЕННЯ

Для експериментальної перевірки було обрано кілька типових сценаріїв, що потребують значних обчислень: сортування великого масиву чисел, застосування графічного фільтру до зображення, генерація простих чисел та стиснення даних. Кожен алгоритм реалізовано двома способами: на чистому JavaScript та на мові C++ з подальшою компіляцією у WebAssembly. Для трансляції C/C++-коду у WASM використано інструментарій Emscripten, який генерує WebAssembly-модуль і "клейовий" JavaScript-код для його завантаження у браузері [5]. Веб-додаток запускали у актуальних версіях браузерів Google Chrome та Mozilla Firefox (які повністю підтримують WASM). Щоб забезпечити коректність вимірювань, виконання кожного сценарію повторювали багаторазово: перші результати ігнорувалися (для прогріву JIT-компілятора JavaScript), а наступні заміри усереднювалися. Час виконання вимірювався засобами високоточних таймерів (Web Performance API) у мілісекундах. Окрім часу виконання, спостерігалися також такі аспекти, як стабільність результатів та навантаження на CPU. Експериментальне середовище уніфіковано: настільний ПК (процесор Intel Core i7, 16 ГБ RAM) під керуванням Windows 11. Таким чином, методика дозволяє безпосередньо порівняти продуктивність JavaScript та WebAssembly в однакових умовах.

5. РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

Отримані результати підтверджують, що використання WebAssembly дає суттєвий приріст швидкодії у порівнянні з традиційним JavaScript. Як показують вимірювання (табл. 1), у всіх протестованих сценаріях WASM-модуль виконував обчислення швидше, ніж аналогічний код на JS. В середньому прискорення склало від 2 до 4 разів, залежно від характеру завдання. Найменший виграш спостерігався для сортування масиву (≈2,7×), що пояснюється ефективністю сучасних JIT-компіляторів JS для цього завдання. Натомість для сценаріїв з інтенсивними обчисленнями – генерації простих чисел та стиснення даних – WebAssembly продемонстрував найбільше прискорення (близько 4× і більше). Застосування графічного фільтру (розмивання зображення 10 МП) показало проміжний результат: WASM приблизно у 2,3 раза швидший за JS у цьому тесті [3]. Дані таблиці 1 ілюструють детальні показники часу виконання для кожного сценарію.

Таблиця 1. Продуктивність сценаріїв

Сценарій	JavaScript	WebAssembly	Прискорення
Сортування масиву	320	120	2,7×
Фільтрація зображення	480	210	2,3×
Генерація простих чисел	710	190	3,7×
Стиснення тексту	1100	240	4,6×

Також на графіку (рис. 1) наочно показано перевагу WebAssembly над JavaScript (сині стовпчики) у всіх дослідних випадках. Видно, що найбільша різниця досягається в задачах, насичених обчисленнями (праворуч): стовпчики WASM суттєво нижчі, що відповідає значно меншому часу виконання.

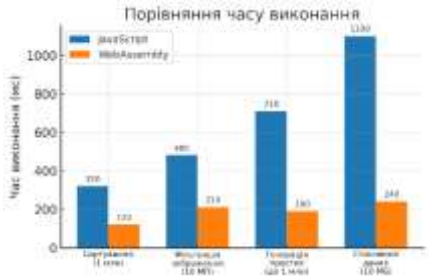

Рисунок 1. Порівняння часу виконання WebAssembly та JavaScript у тестових сценаріях

Наприклад, обчислення простих чисел до заданої межі тривало ~0,19 с у WebAssembly проти ~0,71 с у JS. Аналогічно, стиснення 10 МБ даних виконано за ~0,24 с (WASM) замість ~1,1 с (JS). Отримані цифри узгоджуються з результатами інших дослідників, які також відзначають перевищення швидкодії WASM над JS у типових веб-завданнях. Варто зауважити, що у легких задачах різниця може бути менш відчутною або відсутньою, адже накладні витрати на виклик WebAssembly та ініціалізацію середовища можуть нівелювати виграш. Проте для сценаріїв, котрі вимагають інтенсивних обчислень, використання WebAssembly однозначно покращує продуктивність і забезпечує більш передбачуваний час виконання без “просідань” швидкості, властивих інтерпретованому JS.

6. ВИСНОВКИ

Результати дослідження демонструють доцільність застосування WebAssembly для підвищення продуктивності веб-додатків. WASM дозволяє виконувати критично важливі обчислення у браузері зі швидкодією, близькою до рідного коду, що у наших експериментах дало прискорення від двократного до п’ятикратного. Це відкриває можливості для перенесення у веб-середовище ресурсомістких модулів (наприклад, обробки графіки, шифрування, фізичних двигунів), які раніше були непридатні через повільність JavaScript. WebAssembly органічно співіснує з екосистемою JavaScript, дозволяючи викликати WASM-модулі з коду JS, тож розробники можуть поступово оптимізувати найбільш “важкі” частини додатків. Перспективним напрямом є поява багатопоточності та векторних SIMD-інструкцій у WebAssembly [6], що ще більше збільшить продуктивність в майбутньому. Крім того, розробляються фреймворки (напр. Blazor WebAssembly від Microsoft) та інструменти, які спрощують створення клієнтських веб-застосунків мовами C#/.NET, C++ тощо з використанням WASM [2]. Отже, WebAssembly вже зараз значно розширює межі продуктивності веб-платформи і має всі передумови стати ключовою технологією для високопродуктивних веб-додатків у найближчі роки.

Список літератури

[1] Haas A., Rossberg A., Schuff D.L. та ін. Bringing the Web up to Speed with WebAssembly // Proc. 38th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI). – 2017.

[2] Gallant, G. *WebAssembly in Action*. - Shelter Island, NY : Manning Publications, 2019. - 344 p. - ISBN: 9781617295744.

[3] De Macedo J., Abreu R., Pereira R., Saraiva J. WebAssembly versus JavaScript: Energy and Runtime Performance // Proc. ICT for Sustainability 2022 (IEEE). – 2022. – P. 24– 32.

[4] SkochkoV. I., Ploskyi V. O. Morphogenesis and adjustment of flat rod structures. *USEFUL Online Scientific Journal.* 2018. Vol.2, No 2. P. 8–26. DOI: https://doi.org/10.32557/useful-2-2-2018-0002.

[5] MDN Web Docs. Compiling a new C/C++ module to WebAssembly. URL: https://developer.mozilla.org/en-US/docs/WebAssembly/C_to_wasm (дата звернення: 10.09.2025).

[6] WebAssembly Core Specification : W3C Recommendation, 05.12.2019. URL: https://www.w3.org/TR/wasm-core-1/ (дата звернення: 10.09.2025).