

**"НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО"
(УНИВЕРСИТЕТ ИТМО)**

ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ (РАБОТУ)

Студент Валиуллин Артур Рустемович

(Фамилия, И., О.)

Факультет ПИиКТ Группа Р4107

Направление (специальность) 09.04.04 Программная инженерия

Руководитель Государев И. Б., преподаватель

(Фамилия, И., О., должность)

Дисциплина Инновационные исследования в вебе

Наименование темы: Исследование влияния выбора формата компиляции библиотечных модулей на производительность серверного приложения JavaScript

Задание. Произвести исследование влияния выбора формата компиляции библиотечных модулей на производительность серверного приложения JavaScript

Краткие методические указания (задачи работы)

1. Исследовать инструменты компиляции исходного кода Rust, Go, C++ в модули WebAssembly и DLL;
2. рассмотреть особенности реализации библиотек для сборки в WebAssembly и DLL;
3. исследовать инструменты для измерения производительности функций;
4. реализовать примеры расчета площади криволинейной трапеции методом прямоугольников посредством языков программирования Go, Rust, C++;
5. провести эксперимент с измерением производительности и сделать выводы на основе сравнения результатов.

Содержание пояснительной записки

Оглавление. Введение. Ход выполнения работы – иллюстрированное описание действий и применяемых программ, команд, приёмов, параметров, а также обоснование используемых средств, моделей, параметров. Заключение. Список использованной литературы.

Руководитель

И.Б. Государев

(подпись)

Студент

А.Р. Валиуллин

(подпись)

**"НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИТМО"(УНИВЕРСИТЕТ ИТМО)**

ГРАФИК КУРСОВОГО ПРОЕКТА (РАБОТЫ)

Студент Валиуллин Артур Рустемович

(Фамилия, И., О.)

Факультет ПИиКТ Группа Р4107
Направление (специальность) 09.04.04 Программная
инженерия Руководитель Государев И. Б.
преподаватель

(Фамилия, И., О., должность)

Дисциплина Инновационные исследования в вебе

Наименование темы: Исследование влияния выбора формата компиляции библиотечных модулей на производительность серверного приложения JavaScript

№ п/п	Наименование этапа	Дата завершения		Оценка и подпись руководителя
		Планируемая	Фактическая	
1	Поиск источников, описывающих сравнение производительности модулей WebAssembly и DLL	27.09.2023	27.09.2023	
2	Изучение и анализ источников по WebAssembly и DLL, методов конструирования программ с использованием интерфейса внешних функций	25.10.2023	25.10.2023	
3	Конструирование приложения для измерения производительности модулей, реализованных на выбранных компилируемых языках	15.11.2023	15.11.2023	
4	Оформление отчета	29.11.2023	29.11.2023	

Руководитель

(подпись)

Государев И.Б

Студент

(подпись)

Валиуллин А.Р

"НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО" (УНИВЕРСИТЕТ ИТМО)

АННОТАЦИЯ К КУРСОВОМУ ПРОЕКТУ (РАБОТЕ)

Студент Валиуллин Артур Рустемович
(Фамилия, И., О.)
Факультет ПИиКТ Группа Р4107
Направление (специальность) 09.04.04 Программная инженерия
Руководитель Государев И. Б., преподаватель
(Фамилия, И., О., должность)
Дисциплина Инновационные исследования в вебе

Наименование темы: Исследование влияния выбора формата компиляции библиотечных модулей на производительность серверного приложения JavaScript

ХАРАКТЕРИСТИКА КУРСОВОГО ПРОЕКТА (РАБОТЫ)

1. Цель и задачи работы Определены руководителем Предложены студентом

Цель работы — Исследовать производительность программного кода, скомпилированного в библиотечный модуль динамически подключаемой библиотеки и WebAssembly на примере реализации вычислительной задачи.

Задачи работы:

1. Исследовать инструменты компиляции исходного кода Rust, Go, C++ в модули WebAssembly и DLL;
2. рассмотреть особенности реализации библиотек для сборки в WebAssembly и DLL;
3. исследовать инструменты для измерения производительности функций;
4. реализовать примеры расчета площади криволинейной трапеции методом прямоугольников посредством языков программирования Go, Rust, C++;
5. провести эксперимент с измерением производительности и сделать выводы на основе сравнения результатов.

2. Характер работы Анализ

3. Содержание работы

Исследована производительность программного кода, скомпилированного в библиотечный модуль динамически подключаемой библиотеки и WebAssembly на примере реализации вычислительной задачи расчета площади криволинейной трапеции методом прямоугольников

4. Выводы

Использование библиотечных модулей DLL и WebAssembly компилируемых языков программирования Rust, C++, Go могут увеличить скорость выполнения серверного приложения JavaScript. Для серверного приложения JavaScript эффективно ускоряет выполнения модули динамически подключаемых библиотек. Использование библиотечных модулей (WebAssembly и DLL), реализованных средствами языка программирования Go, показывает наименьшую эффективность. Для WebAssembly Rust показывает лучшую производительность.

Руководитель _____
(подпись)

И.Б. Государев

Студент _____
(подпись)

А.Р. Валиуллин

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский университет ИТМО»
(Университет ИТМО)

Факультет программной инженерии и компьютерной техники

Направление (специальность) 09.04.04 Программная инженерия

Специализация Веб-технологии

Дисциплина — Инновационные исследования в вебе

Курсовой проект (работа)

ТЕМА: Исследование влияния выбора формата компиляции библиотечных модулей на производительность серверного приложения JavaScript

ВЫПОЛНИЛА

Студент группы

P4107

№ группы

Валиуллин А.Р.

ФИО

ПРОВЕРИЛ

преподаватель

ученая степень, должность

Государев И. Б

ФИО

САНКТ-ПЕТЕРБУРГ 2023 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 ОБЗОР МОДУЛЕЙ DLL И WEBASSEMBLY	4
2 МЕТОДЫ ИЗМЕРЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ	5
3 ОСОБЕННОСТИ КОНСТРУИРОВАНИЕ И СБОРКА	6
4 ПРОИЗВОДИТЕЛЬНОСТЬ	9
ЗАКЛЮЧЕНИЕ	15

ВВЕДЕНИЕ

Для вызова функций сторонних библиотек компилируемых языков программирования для серверных приложения реализованных языке JavaScript могут использоваться два подхода. В первом подходе исходный код библиотеки может быть скомпилирован в формат динамически подключаемой библиотеки и функции этой библиотеки могут быть вызваны в приложении посредством FFI (интерфейса внешних функций). Бинарный код динамически подключаемой библиотеки исполняется операционной системой. Во втором подходе, исходный код библиотеки может быть скомпилирован в бинарный формат WebAssembly, который может исполняться платформой JavaScript – браузером, серверными платформами NodeJS, Bun.

В качестве компилируемых языков в рамках курсовой работы используются языки Go, Rust, C++. В качестве платформы JavaScript используется Bun.

Целью исследования в данной курсовой работе является сравнительный анализ производительности программного кода, скомпилированного в библиотечный модуль динамически подключаемой библиотеки и WebAssembly на примере реализации вычислительной задачи расчета площади криволинейной трапеции методом прямоугольников.

Для достижения цели исследования определены следующие задачи:

1. Исследовать инструменты компиляции исходного кода Rust, Go, C++ в модули WebAssembly и DLL;
2. рассмотреть особенности реализации библиотек для сборки в WebAssembly и DLL;
3. исследовать инструменты для измерения производительности функций;
4. реализовать примеры расчета площади криволинейной трапеции методом прямоугольников посредством языков программирования Go, Rust, C++;
5. провести эксперимент с измерением производительности и сделать выводы на основе сравнения результатов.

1 ОБЗОР МОДУЛЕЙ DLL И WEBASSEMBLY

Поскольку выполнение программного кода, скомпилированного в формат динамически подключаемых библиотек, происходит операционной системой, приложения реализованные на JavaScript использующие интерфейс внешних функций являются платформозависимыми и могут успешно работать только на операционной системе, для которой была собрана динамически подключаемая библиотека. Библиотека, скомпилированная в бинарный формат WebAssembly, исполняется платформой JavaScript, что позволяет выполнять вызов функций таких библиотек независимо от операционной системы, любой платформой JavaScript в том числе и браузерами.

Динамически подключаемые библиотеки используются для эффективной организации памяти, путем создания одного бинарного библиотечного модуля для нескольких приложений.

WebAssembly используется для ускорения работы приложения JavaScript за счет выполнения низкоуровневого байт-кода [1]. Библиотечные модули WebAssembly могут быть реализованы на языках программирования помимо JavaScript, для которых представлены соответствующие компиляторы, такие как [2, 3]:

- C/C++ - Emscripten;
- Rust;
- Java – TeaVV и JWebAssembly;
- Kotlin – Kotlin/Native через LLVM;
- Go;
- C# - Blazer и Uno Platform;
- TypeScript – AssemblyScript.

2 МЕТОДЫ ИЗМЕРЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ

Замеры производительности исполняемого кода могут быть выполнены посредством интерфейса Performance. Этот интерфейс представляется движком браузера, а также платформой Bun. Метод `performance.now()` возвращает временную метку в миллисекундах [4]. Для измерения времени выполнения кода, метод необходимо применить до и после тестируемого фрагмента и вычесть разницу из полученных временных меток (рисунок 1).

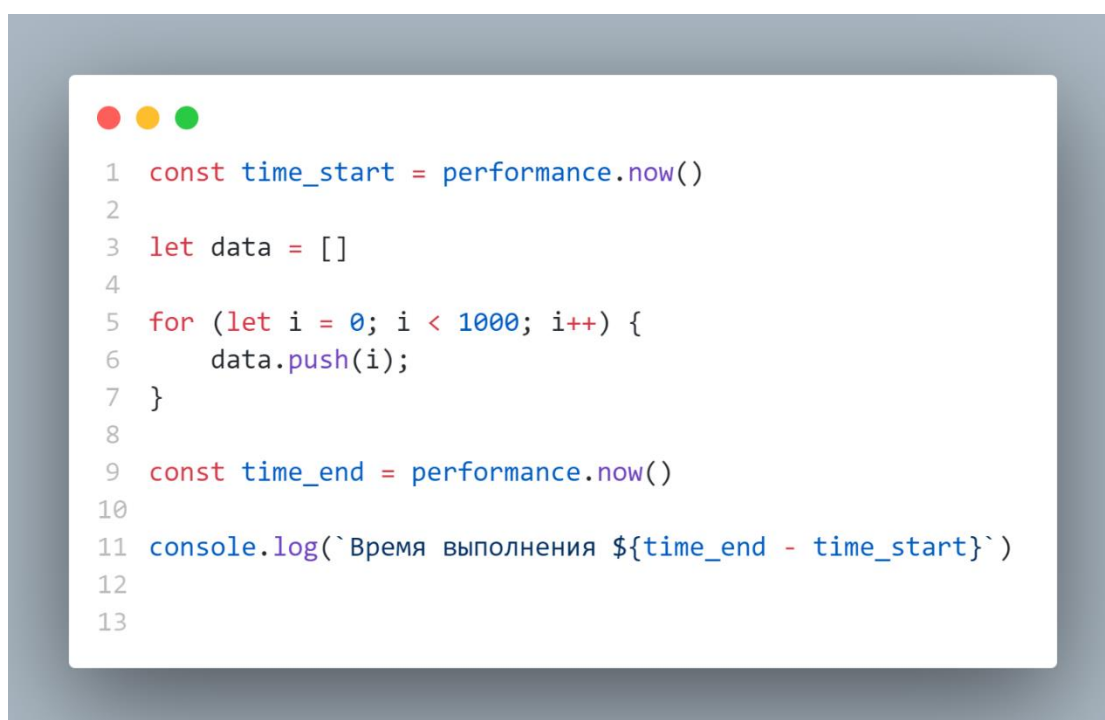



Рисунок 1 - Пример измерения скорости участка кода

3 ОСОБЕННОСТИ КОНСТРУИРОВАНИЕ И СБОРКА

Для создания динамически подключаемой библиотеки на языке программирования C++ были использованы программные средства LLVM, Clang [5] с использованием уровня оптимизации (-O2) [6], увеличивающих скорость исполнения кода, который по умолчанию используют средства компиляции Rust, Go в сборке в конфигурации «выпуск».

Для сборки библиотек представлены инструменты rustc, go. На рисунке 2 представлен сценарий сборки модулей для C++, Rust, Go.

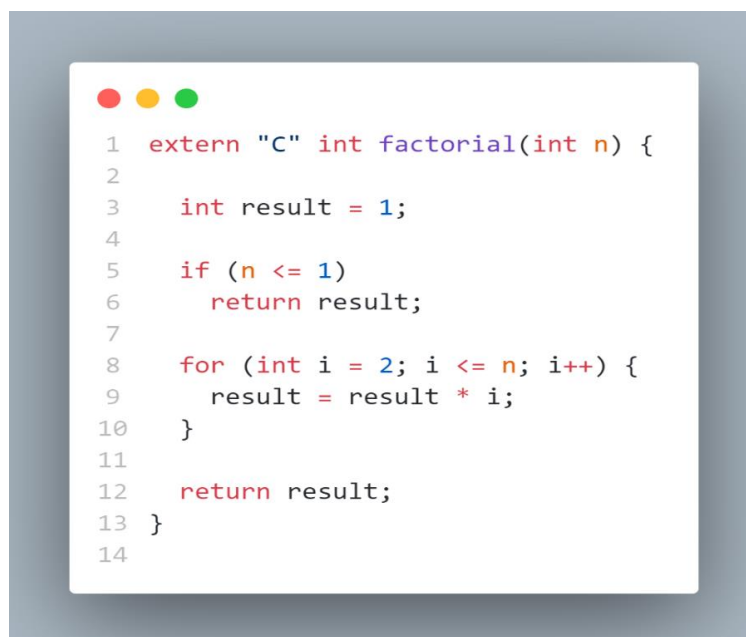


```
1 #!/usr/bin/env sh
2
3 # Очистка папки перед сборкой
4 rm -rf $PWD/out
5 mkdir -p $PWD/out
6
7 # Сборка динамически подключаемой библиотеки для ЯП C++
8 clang -c $PWD/src/lib.cpp -O2 -o $PWD/out/lib_cpp.o
9 clang -shared -o $PWD/out/lib_cpp.so $PWD/out/lib_cpp.o
10
11 # Сборка динамически подключаемой библиотеки для ЯП Rust
12 rustc --crate-type cdylib -O $PWD/src/lib.rs -o $PWD/out/lib_rs.so
13
14
15 # Сборка динамически подключаемой библиотеки для ЯП Go
16 go build -o $PWD/out/lib_go.so -buildmode=c-shared $PWD/src/lib.go
17
```

Рисунок 2 - Сценарий сборки динамически подключаемых библиотек

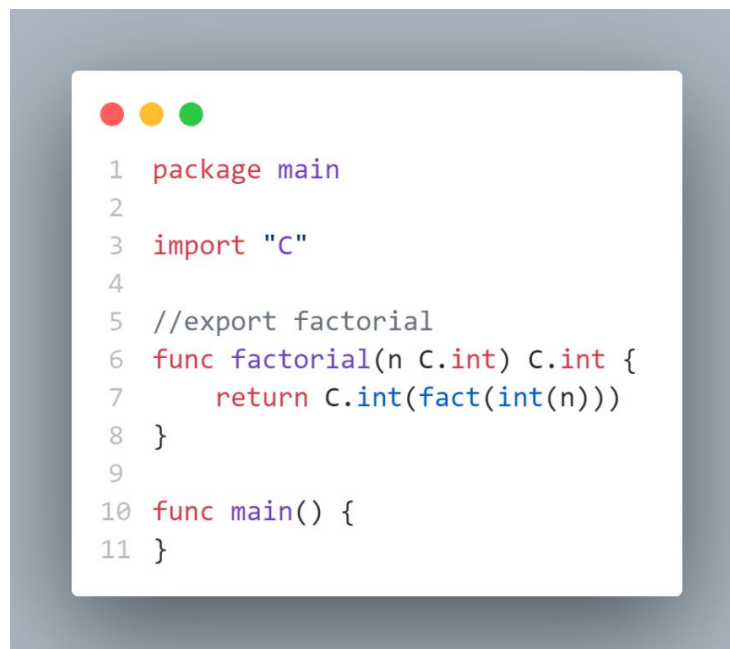
Для того чтобы функции подключаемых модулей было возможно вызывать из JavaScript для компилируемых языков необходимо прописать в определении функции директивы, которые делают эти функции видимыми и не искажают имена вызываемых функций. Для C++ и Rust это директива «extern C» [7], для Go

используются служебные комментарии и модуль cgo. В модуле Go обязательно должна быть определена точка входа. Примеры представлены на рисунках 3, 4, 5.

A screenshot of a code editor with a white background and a grey border. At the top left, there are three colored circles: red, yellow, and green. The code is written in C and is a factorial function. It starts with a line number 1 and ends with 14. The code is as follows:

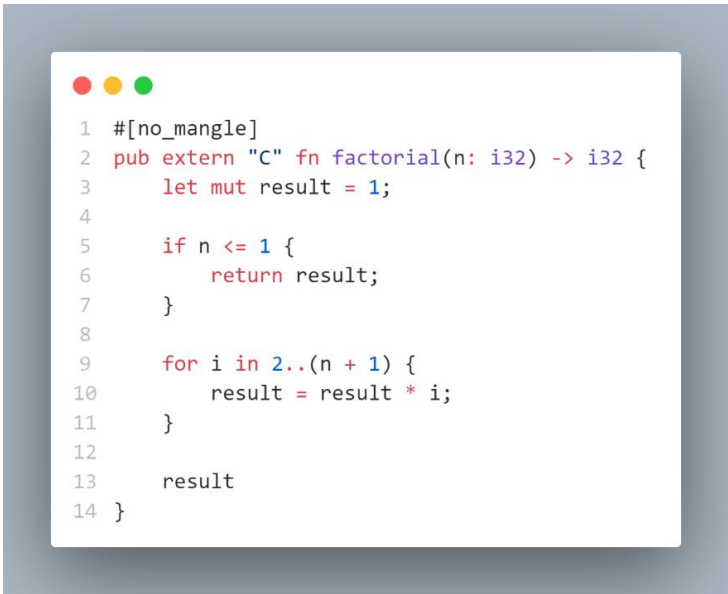
```
1 extern "C" int factorial(int n) {  
2  
3     int result = 1;  
4  
5     if (n <= 1)  
6         return result;  
7  
8     for (int i = 2; i <= n; i++) {  
9         result = result * i;  
10    }  
11  
12    return result;  
13 }  
14
```

Рисунок 3 - Пример функции с директивой для вызова внешним кодом C++

A screenshot of a code editor with a white background and a grey border. At the top left, there are three colored circles: red, yellow, and green. The code is written in Go and is a factorial function. It starts with a line number 1 and ends with 11. The code is as follows:

```
1 package main  
2  
3 import "C"  
4  
5 //export factorial  
6 func factorial(n C.int) C.int {  
7     return C.int(fact(int(n)))  
8 }  
9  
10 func main() {  
11 }
```

Рисунок 4 - Пример функции с директивой для вызова внешним кодом Go




```

1  #[no_mangle]
2  pub extern "C" fn factorial(n: i32) -> i32 {
3      let mut result = 1;
4
5      if n <= 1 {
6          return result;
7      }
8
9      for i in 2..(n + 1) {
10         result = result * i;
11     }
12
13     result
14 }

```

Рисунок 5 - Пример функции с директивой для вызова внешним кодом Rust

Для сборки библиотек WebAssembly представлены следующие средства. Для C++ - Emscripten [8], Rust – wasm-pack [9]. В Go средства сборки модулей WebAssembly входят в состав стандартных средств. Пример сценария сборки представлен на рисунке 6.



```

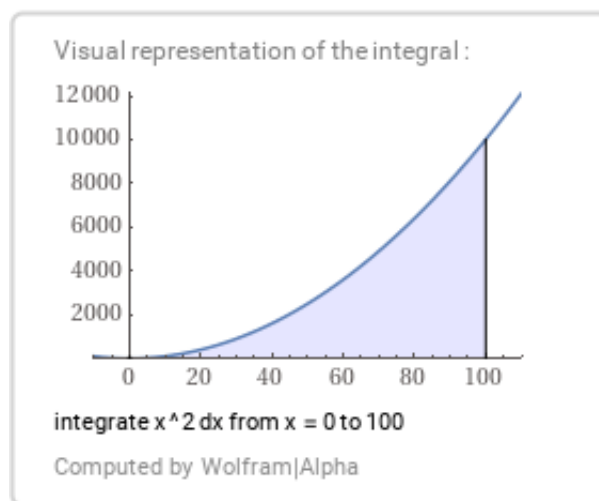
1  #!/usr/bin/env sh
2
3  # Очистка
4  rm -rf $PWD/out
5  rm -rf $PWD/pkg
6  mkdir -p $PWD/out
7
8  # Сборка библиотеки C++ для WebAssembly
9  em++ $PWD/src/lib.cpp -s WASM=1 -s EXPORTED_FUNCTIONS=["'_x2Integrate'"] \
10     -s EXPORTED_RUNTIME_METHODS=ccall -s MODULARIZE -o $PWD/out/lib_cpp.out.js
11
12 # Сборка библиотеки Go для WebAssembly
13 export GOARCH=wasm
14 export GOOS=js
15 go build -o $PWD/out/lib_go.out.wasm $PWD/src/lib.go
16
17 # Сборка библиотеки Rust для WebAssembly
18 wasm-pack build --target nodejs
19

```

Рисунок 6 - Сценарий сборки модулей WebAssembly

4 ПРОИЗВОДИТЕЛЬНОСТЬ

В качестве задачи для расчета в подключаемых модулях была выбрана задача численного интегрирования методом прямоугольников [10]. На рисунке 7 представлен расчет в среде WolframAlpha. На рисунке 8 представлен исходный код решения задачи на C++.



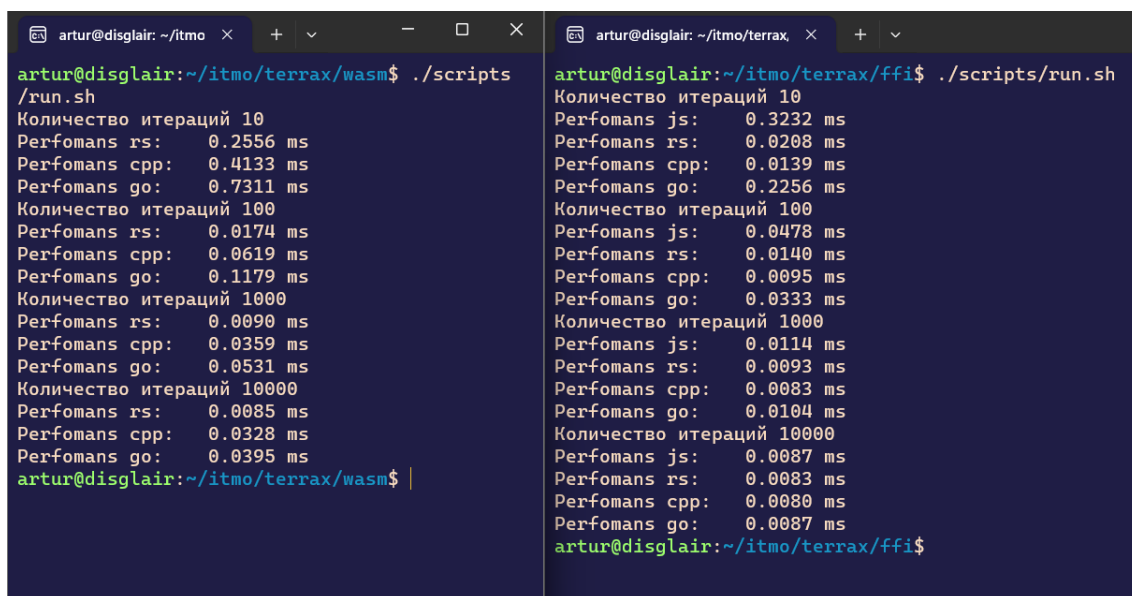
$$\int_0^{100} x^2 dx = \frac{1\,000\,000}{3} \approx 3.3333 \times 10^5$$

Рисунок 7 - Площадь под кривой

```
1 inline double f(double x) { return x * x; }
2
3 extern "C" double x2Integrate(double xmin, double xmax, int intervals_count) {
4     double dx = (xmax - xmin) / intervals_count;
5     double total = 0.0;
6     double x = xmin;
7     for (int i = 1; i < intervals_count; i++) {
8         total = total + dx * (f(x) + f(x + dx)) / 2.0;
9         x = x + dx;
10    }
11    return total;
12 }
13
```

Рисунок 8 - Исходный код расчета на C++

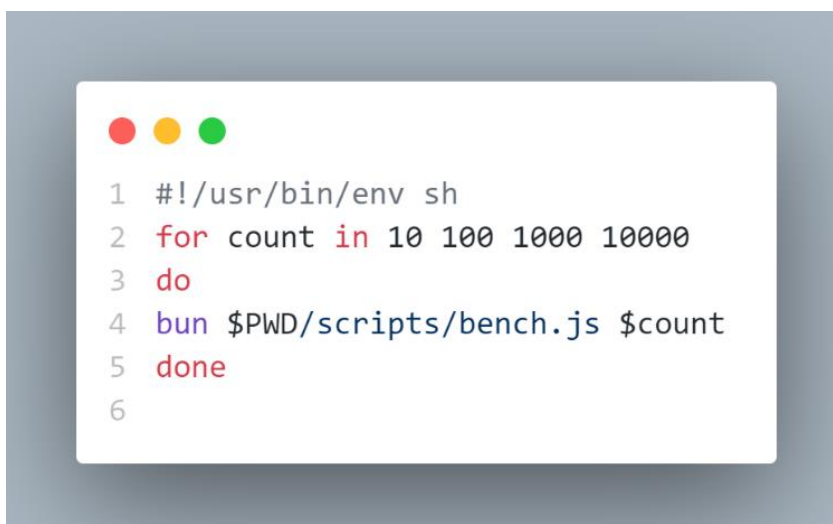
Для модулей на языках Rust, Go, JavaScript реализованы аналогичные решения. На рисунке 9 представлен вывод измерения времени выполнения функции в эксперименте. Для измерения времени были реализованы два сценария. В первом, функции многократно вызывались внутри приложения JavaScript. Для каждой функции были выполнены замеры среднего времени выполнения по 10, 100, 1000, 10000 вызовов. Посредством сценария bash, представленном на рисунке 10. Во втором функции вызывались один раз в рамках приложения.



```
artur@disglair: ~/itmo/terrax/wasm$ ./scripts/run.sh
Количество итераций 10
Perfomans rs: 0.2556 ms
Perfomans cpp: 0.4133 ms
Perfomans go: 0.7311 ms
Количество итераций 100
Perfomans rs: 0.0174 ms
Perfomans cpp: 0.0619 ms
Perfomans go: 0.1179 ms
Количество итераций 1000
Perfomans rs: 0.0090 ms
Perfomans cpp: 0.0359 ms
Perfomans go: 0.0531 ms
Количество итераций 10000
Perfomans rs: 0.0085 ms
Perfomans cpp: 0.0328 ms
Perfomans go: 0.0395 ms
artur@disglair:~/itmo/terrax/wasm$ |

artur@disglair: ~/itmo/terrax/ffi$ ./scripts/run.sh
Количество итераций 10
Perfomans js: 0.3232 ms
Perfomans rs: 0.0208 ms
Perfomans cpp: 0.0139 ms
Perfomans go: 0.2256 ms
Количество итераций 100
Perfomans js: 0.0478 ms
Perfomans rs: 0.0140 ms
Perfomans cpp: 0.0095 ms
Perfomans go: 0.0333 ms
Количество итераций 1000
Perfomans js: 0.0114 ms
Perfomans rs: 0.0093 ms
Perfomans cpp: 0.0083 ms
Perfomans go: 0.0104 ms
Количество итераций 10000
Perfomans js: 0.0087 ms
Perfomans rs: 0.0083 ms
Perfomans cpp: 0.0080 ms
Perfomans go: 0.0087 ms
artur@disglair:~/itmo/terrax/ffi$
```

Рисунок 9 – Вывод результатов измерения



```
1 #!/usr/bin/env sh
2 for count in 10 100 1000 10000
3 do
4 bun $PWD/scripts/bench.js $count
5 done
6
```

Рисунок 10 – Сценарий баш для многократного вызова

Результаты измерения времени выполнения программного кода, собранном в бинарном формате WebAssembly представлены в таблице 1 и рисунке 11.

Таблица 1 – Время выполнения модуля собранном в формате WebAssembly (мс)

	Номер эксперимента				
Количество итераций	1	2	3	4	Среднее время
Rust					
10	0,1136	0,0787	0,0670	0,0668	0,0815
100	0,0240	0,0175	0,0170	0,0173	0,0190
1000	0,0970	0,0092	0,0094	0,0092	0,0312
10000	0,0860	0,0085	0,0084	0,0084	0,0278
C++					
10	0,2793	0,2360	0,3086	0,2973	0,2803
100	0,0545	0,0527	0,0575	0,0525	0,0543
1000	0,0353	0,0357	0,0351	0,0349	0,0353
10000	0,0336	0,0330	0,0337	0,0335	0,0335
Go					
10	0,6150	0,6057	0,6380	0,6293	0,6220
100	0,1152	0,1202	0,1263	0,1156	0,1193
1000	0,0534	0,0552	0,0540	0,0532	0,0540
10000	0,0413	0,0403	0,0414	0,0426	0,0414

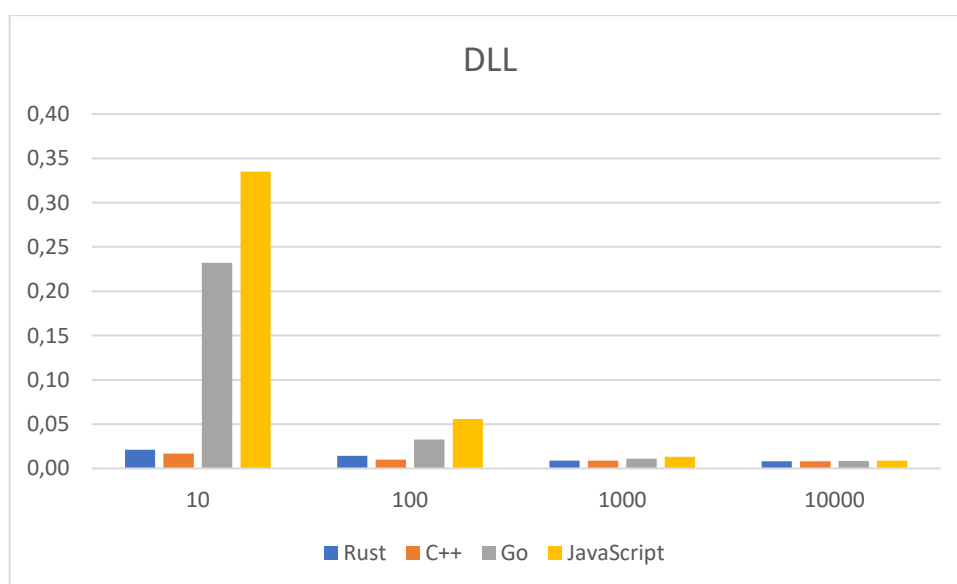


Рисунок 11– Сравнение скорости выполнения для DLL

Результаты измерения времени выполнения программного кода, собранном в бинарном формате DLL представлены в таблице 2 и рисунке 12.

Таблица 2 – Время выполнения модуля собранном в формате DLL (мс)

	Номер эксперимента				
Количество итераций	1	2	3	4	Среднее время, мс
Rust					
10	0,0214	0,0220	0,0205	0,0203	0,0211
100	0,0157	0,0140	0,0154	0,0122	0,0143
1000	0,0090	0,0090	0,0090	0,0087	0,0089
10000	0,0082	0,0081	0,0082	0,0081	0,0082
C++					
10	0,0144	0,0223	0,0171	0,0136	0,0169
100	0,0105	0,0099	0,0099	0,0096	0,0100
1000	0,0083	0,0084	0,0091	0,0098	0,0089
10000	0,0081	0,0081	0,0081	0,0081	0,0081
Go					
10	0,1752	0,2432	0,3485	0,1610	0,2320
100	0,0247	0,0353	0,0388	0,0321	0,0327
1000	0,0102	0,0112	0,0119	0,0105	0,0110
10000	0,0085	0,0088	0,0086	0,0085	0,0086
JavaScript					
10	0,3269	0,3135	0,3905	0,3094	0,3351
100	0,0502	0,0492	0,0487	0,0757	0,0560
1000	0,0128	0,0124	0,0146	0,0126	0,0131
10000	0,0086	0,0087	0,0088	0,0086	0,0087

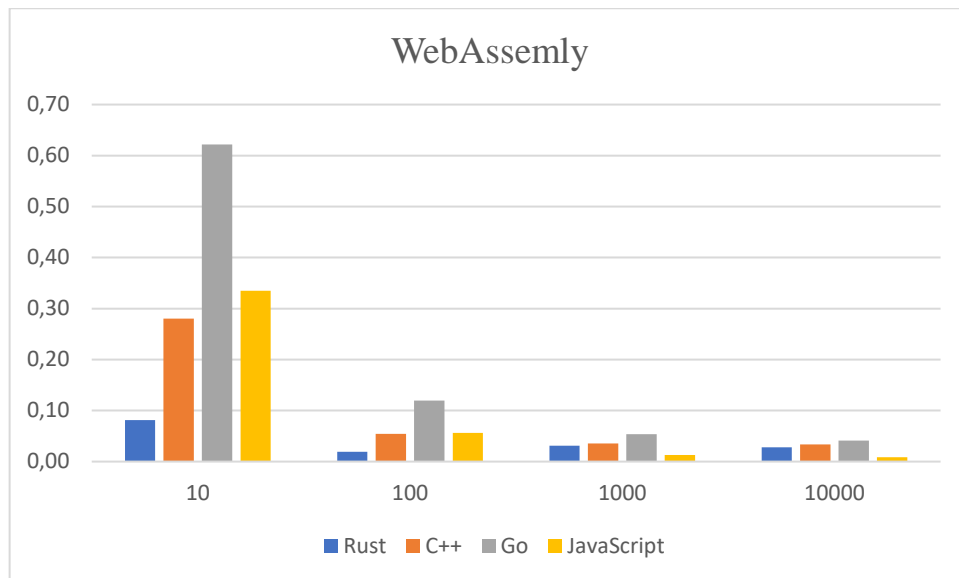


Рисунок 12 - Сравнение скорости выполнения для WebAssembly

Сравнение скорости однократного выполнения функции в рамках одного процесса представлены в таблице 3, рисунках 13, 14.

Таблица 3 – Сравнительное время выполнения WebAssembly и DLL (мс)

	Rust	C++	Go	JavaScript
WebAssembly	0,5614	0,9201	0,9815	1,1413
DLL	0,1325	0,0256	2,5305	1,1413

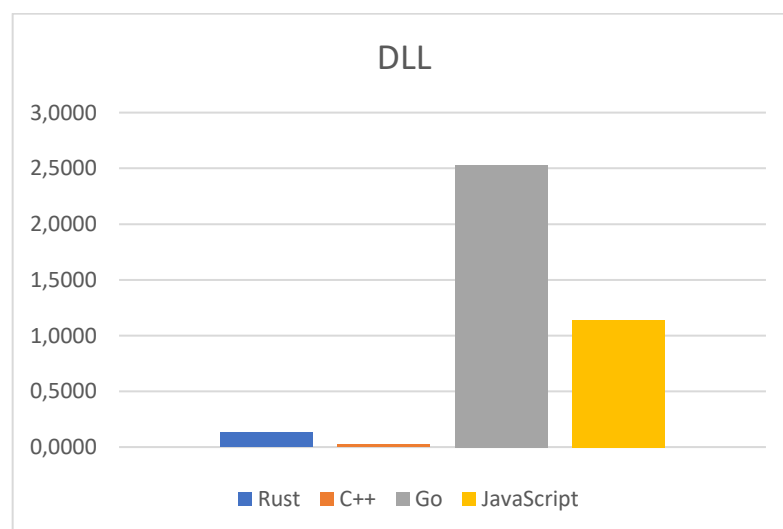


Рисунок 13 - Сравнение скорости выполнения функций DLL.

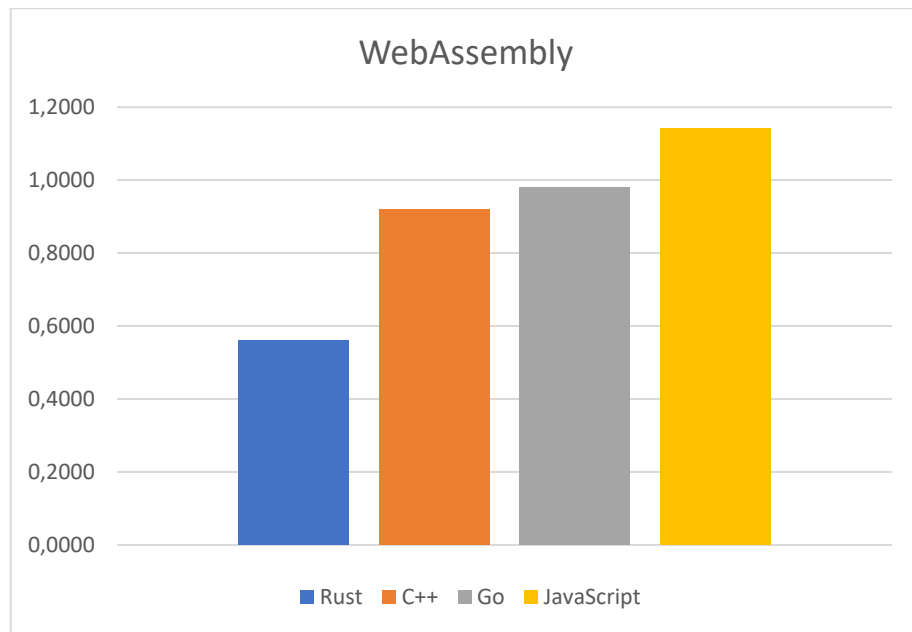


Рисунок 14 - Сравнение скорости выполнения функций WebAssembly

ЗАКЛЮЧЕНИЕ

Использование библиотечных модулей DLL и WebAssembly компилируемых языков программирования Rust, C++, Go могут увеличить скорость выполнения серверного приложения JavaScript. Для серверного приложения JavaScript эффективно ускоряет выполнения модули динамически подключаемых библиотек. Использование библиотечных модулей (WebAssembly и DLL), реализованных средствами языка программирования Go, показывает наименьшую эффективность. Для WebAssembly Rust показывает лучшую производительность.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. 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>
2. Sletten B. WebAssembly: The Definitive Guide. – "O'Reilly Media, Inc.", 2021.
3. Балабаш М. А. WebAssembly–путь к новым горизонтам производительности в Web // Семантические фреймы: классификаторы и квалификаторы. 2017. С. 214-216. URL: <https://elib.psu.by/bitstream/123456789/35428/1/214-216.pdf>
4. 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.
5. Clang: a C language family frontend for LLVM / [Электронный ресурс] //: [сайт]. — URL: 5. <https://clang.llvm.org> (дата обращения: 28.11.2023).
6. Microsoft Learn / [Электронный ресурс] //: [сайт]. — URL: <https://learn.microsoft.com/ru-ru/cpp/build/reference/o1-o2-minimize-size-maximize-speed?view=msvc-170> (дата обращения: 28.11.2023).
7. Объявления внешних функций / [Электронный ресурс] // : [сайт]. — URL: <https://learn.microsoft.com/ru-ru/cpp/cpp/extern-cpp?view=msvc-170#extern-c-and-extern-c-function-declarations> (дата обращения: 28.11.2023).
8. Introducing Emscripten / [Электронный ресурс] // : [сайт]. — URL: https://emscripten.org/docs/introducing_emscripten/index.html (дата обращения: 28.11.2023).
9. Learn Rust and WebAssembly / [Электронный ресурс] // : [сайт]. — URL: <https://rustwasm.github.io/> (дата обращения: 28.11.2023).

10. Шавкатбекова Ш. Ш. Реализация методов численного интегрирования и дифференцирования в системе Маткад //Kazakhstan Science Journal. – 2019. – Т. 2. – №. 1. – С. 57-63.