WEBASSEMBLY

А.И. Рудаков
Научный руководитель: Е.А. Кочегурова
Томский Политехнический Университет
E-mail: alanmelone@gmail.com

Введение

В настоящее время потребность выполнения ряда задач в брузере возрастает. Среди таких задач можно встретить запуск 3D-игр, виртуальная и дополненная реальность, компьютерное зрение, рендеринг больших изображений и видео. Данные сценарии достаточно нагружены и существующие технологии уже не способны решать проблемы исполнения задач с достаточной производительностью, близкой к процессорной. Для решения проблемы производительности был разработан формат байт-игр, кода, который быстрее исполняется в браузере.

WebAssembly (WASM) – это новый открытый формат байт-игр, кода, исполняемого современными браузерами. Технология WebAssembly создается как открытый стандарт внутри W3C WebAssembly Community Group [1].

Данная технология призвана решить основную проблему исполнения кода в браузере – производительность вычислений. Но данная проблема не может быть решена без удолетворения ряда требований, поскольку основной средой выполнения является браузер [2]. Основные из них следующие:

1. **Zero configuration** – требование решения «из коробки», ничего кроме браузера.
2. **Безопасно** – новая технология не должна создавать новых угроз.
3. **Кроссплатформенно** – браузеры работают на всех основных процессорах, включая мобильные платформы.
4. **Удобство для разработчиков** – удобные средства разработки и отладки.

Также WebAssembly позволяет запускать приложения написанные на других языках, код которых скомпилирован в WASM байт-код.

Виртуальная машина WASM

WebAssembly исполняется JavaScript-движком (V8, Tamarin и др.) в виртуальной стековой машине. По данным сайта https://caniuse.com/ на январь 2020 WASM поддерживает 88.53% браузеров.

Виртуальная машина имеет четыре типа данных: i32, i64, f32, f64. Также в неё включены операции с памятью, со стеком, ветвления и конвертации. WASM имеет плоскую модель памяти. Под память выделяется единый блок, размер которого кратен 64 КБ. Здесь находится код, данные, константы, глобальные переменные, абстрактный тип данных стек растущий вниз и структура данных куча, растущая вверх. Можно сделать так, чтобы куча автоматически увеличивалась при необходимости, при этом блок памяти расширяется на размер кратный 64 КБ. По соображениям безопасности указатели не используются, вместо них присутствуют индексы. Индекс 32-разрядный, поэтому адресуется до 4 ГБ памяти. Вся память WebAssembly полностью доступна из JavaScript, причём как на чтение, так и на запись [2]. На рисунке 1 представлена схема памяти виртуальной машины.

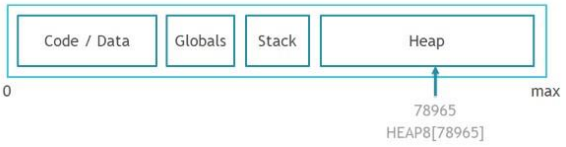

Рис. 1. Архитектура памяти виртуальной машины WASM

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. Общая схема выполнения WASM модуля представлена на рисунке 2.

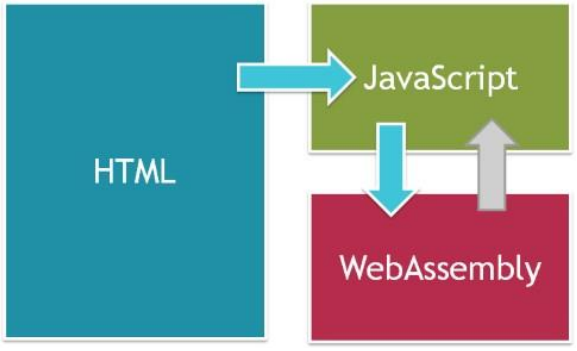

Рис. 2. Общая схема выполнения WebAssembly в браузере

WASM модуль может быть загружен как локально, так и получен через Интернет.

Создание WebAssembly модуля

Так как WebAssembly не язык программирования, модули для него обычно не пишутся, а компилируются из других языков программирования. Для компиляции из других языков существует проект LLVM, который состоит из набора компиляторов для языков программирования, системы оптимизации,

интерпретации и компиляции в машинный код [3]. Для LLVM написано множество компиляторов с языка программирования в промежуточное представление кода, так называемые «фронтенд» компиляторы. Затем промежуточное представление кода преобразуется в код для конкретной платформы (x86/x86-64, ARM и т.д), так называемые «бэкенд» компиляторы. Официальным компилятором WASM для языка C/C++ является Emscripten. Данный компилятор предоставляет свой LLVM «бэкенд» для сборки в WebAssembly.

Emscripten позволяет получить из исходного C/C++ кода код на JavaScript. При этом компилятор сгенерирует WASM модуль, который будет вызываться сгенерированным JavaScript кодом.

Практическое применение

Для демонстрации возможности запуска программ, написанных на языке C/C++ в браузере, была взята простая программа, реализующая класс вектора с тремя компонентами: x, y, z. С помощью Emscripten был скомпилирован WASM байт-код, JavaScript код и HTML шаблон по умолчанию. Результат работы программы представлен на рисунке 3.

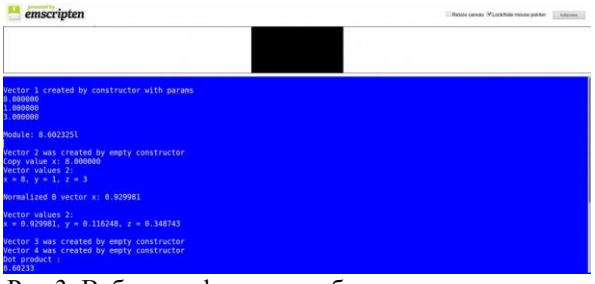

Рис.3. Веб-интерфес для отображения результатов программы

На рисунке изображена консоль ввода-вывода. Сверху отображается canvas элемент, который отвечает за отрисовку графики.

Данная демонстрация не отражает полностью возможность WebAssembly, так как нет высокой вычислительной нагрузки.

В качестве демонстрации возможностей применения этой технологии можно взять портированную игру Doom 3, на движке idTech 4 от d3wasm. Данный порт достаточно функционален и работоспособен, но только для настольных систем. Что касается запуска на мобильных устройствах, то авторы пишут, что работа на мобильном устройстве не удобна в связи с управлением и плохой производительностью [4]. Судя по данному примеру технология ещё не готова войти массово в браузеры всех устройств и использовать все возможности процессоров.

Также в качестве примера можно привести сервис для проектирования пользовательского интерфеса Figma. Как пишут разработчики, внедрение WebAssembly в их приложение позволило ускорить загрузку рендера их пользовательского интерфейса в три раза [5].

Также есть опыт использования у веб-приложения fastq.bio. В этом веб-приложении увеличили количество операций чтения в секунду в двадцать раз. Но это не значит, что производительность приложения повысилась в такое же количество раз. Из-за взаимодействия JavaScript и WebAssembly можно добиться максимального увеличения производительности от 20% до двух раз [6].

Заключение

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.

Список использованной литературы

1. WebAssembly [Электронный ресурс] | MDN – URL: https://developer.mozilla.org/ru/docs/WebAssembly (Дата обращения: 21.01.2020)
2. WebAssembly: что и как | Хабр – URL: https://habr.com/ru/post/475778 (Дата обращения: 21.01.2020)
3. LLVM | Wikipedia – URL: https://ru.wikipedia.org/wiki/LLVM (Дата обращения: 22.01.2020)
4. D3Wasm: a port of idTech 4 | Doom 3D-игр, engine to WebAssembly – URL: http://www.continuation-игр,labs.com/projects/d3wasm/ (Дата обращения: 22.01.2020)
5. Figma is powered by WebAssembly | Figma– URL:https://www.figma.com/blog/webassembly-cut-figmas-load-time-by-3x/(Дата обращения: 22.01.2020)
6. How we used WebAssembly to speed up our web app by 20x | Smashing Magazine – URL: https://www.smashingmagazine.com/2019/04 /webassembly-speed-web-app/ (Дата обращения: 23.01.2020)