코드 단위별 웹 어셈블리와 자바스크립트의 성능 분석

김아영, 박태준*
전남대학교(학부생), 전남대학교(교수)*
cpjvh@jnu.ac.kr, taejune.park@jnu.ac.kr

Performance analysis of WebAssembly and JavaScript by code unit

Ayoung Kim, Taejune Park*
Chonnam National Univ.

요 약

최근 웹 시장이 성장하면서 고성능을 요구하는 AR/VR, 3D 게임 등이 웹 환경에서 구현되기 시작하였다. 하지만 기존의 웹 표준 언어인 자바스크립트(javascript, js)는 C/C++ 같은 네이티브(native) 언어보다 높은 연산을 수행하기에는 성능이 부족하다는 한계가 있었다. 이런 성능 문제를 극복하기 위해서 저수준 바이트코드인 웹 어셈블리(WebAssembly, wasm)가 등장하였다. 웹 어셈블리는 자바스크립트보다 처리 과정이 짧고 바이너리 수준에서 작동되기 때문에 더 좋은 성능을 나타낸다고 알려져 있다. 따라서 본 논문은 가상화 환경에서 엠스크립튼(emscripten)을 사용한 웹 어셈블리가 자바스크립트보다 얼마나 더 나은 성능을 지니는지 동일한 문맥(context)의 코드 단위별 사례들을 두 가지 버전으로 작성하고 측정 및 비교한 결과에 대해 알아보았다.

Ⅰ. 서 론

자바스크립트는 네이티브 언어보다 성능이 부족하다는 한계가 존재한다. 이런 문제를 해결하기 위해 웹 어셈블리라는 새로운 기술이 등장하였다. 웹 어셈블리는 네이티브 언어와 유사한 성능을 나타내고 있고 [1,2,3] 바이너리 수준에서 작동되기 때문에 자바스크립트보다 더 좋은 성능을 나타낸다고 알려져 있다. 또한 웹 어셈블리는 자바스크립트에 존재하는 구문해석과 재-최적화, 가비지 컬렉션(garbage collection)이 처리 과정에 없어 성능 처리 면에서 자바스크립트보다 더 뛰어나다.

본 논문에서는 웹 어셈블리가 자바스크립트보다 얼마나 더 나은 성능을 나타내는지 알아보기 위해 동일한 문맥의 코드 단위별로 실험하여 비교해 보고자 한다. 웹 어셈블리 파일은 그림 1처럼 네이티브 언어로 구성되어있는 코드를 엠스크립튼 컴파일러 [4,5]를 통해 웹 어셈블리 파일과 자바스크립트인 glue code, html을 생성하였다. 엠스크립튼을 통해 만들어진 웹 어셈블리와 자바스크립트의 7가지 코드를 동일한 문맥의 코드로 각각 작성하여 실험을 진행하였다.

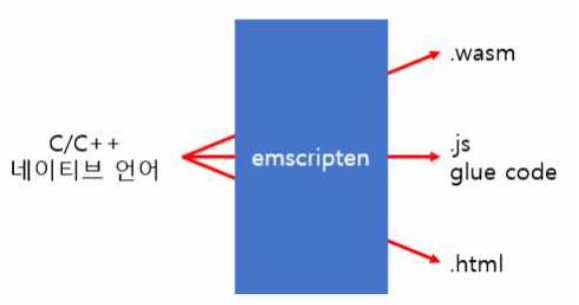

그림 1. emscripten complier

Ⅱ. 본론

1. 용어 및 개념

1) 웹 어셈블리

웹 어셈블리는 기존의 자바스크립트의 성능 문제를 해결하기 위해 등장한 저수준 바이트코드이다. C/C++, Rust 등과 같은 다양한 네이티브 언어로 작성된 코드를 컴파일하여 이진 바이너리 파일로 생성된다. 이런 바이너리 파일은 웹에 빠르게 로드하여 네이티브 언어와 유사한 속도로 실행된다. 2019년 웹 어셈블리는 웹 표준을 관리하는 W3C(World Wide Web Consortium)에서 html. css, 자바스크립트에 이어 웹의 4번째 언어로 공식 권고하였다 [6].

2) 자바스크립트

자바스크립트는 런타임 도중에 컴파일 과정을 수행하는 동적 프로그래밍 언어로 변수의 타입을 지정할 필요가 없고 타입이 런타임 과정에서 변한다는 특징을 가지고 있다. 동적 프로그래밍 언어는 컴파일러가 컴파일하기 전에 타입을 예측할 시간이 필요하고 예측한 타입이 틀렸을 경우 재-최적화하는 단계가 추가로 필요하다. 때문에 자바스크립트는 기존의 네이티브 언어들보다 컴파일 시간이 느리다.

2. 성능 측정 및 비교

자바스크립트와 웹 어셈블리의 성능을 직접 비교해보기 위해, 본 연구에서는 동일한 문맥에 해당하는 코드 사례들을 자바스크립트와 웹 어셈블리 두 가지 버전으로 작성하고, 그 실행 시간을 가상화 환경인 표1의 환경에서 비교해보았다. C언어에서는 clock_gettime() 함수와 자바스크립트에서 nano-time 패키지 [7]를 이용하였고 결과는 node.js를 통해 측정하였다. 실험은 10회 반복 실행한 결과에 대한 평균을 측정하였으며, 아래 표2는 그 결과를 정리한 것이다.

표1. 실험 환경

분류	사양
CPU	A virtual machine with 2 cores on VMware ESXi and Intel(R) Xeon(R) CPU E5-2640
OS	Debian GNU/Linux 9(64bit)
Memory	4GB on VM
gcc	11.3.0 (Debian 11.3.0-1)
Emscripten	3.1.29

표2. 7개의 사례에 따른 10회 실행 값의 평균

사례	js (micro)	wasm (micro)	x N
#1 (조건문)	5097.09	5756.27	1.12
#2 (거품 정렬)	511.718	75.2164	6.8
#3 (함수 호출 반복문 - 100회)	6265.27	24.9397	251.21
#4 (정수형 변수 반복문 - 100회)	101.043	9.1494	11.04
#5 (실수형 변수 반복문 - 100회)	126.592	11.4613	11.04
#6 (정수형 변수 반복문 - 10,000회)	342.63	44.5118	7.69
#7 (정수형 변수 반복문 - 100,000회)	2066.51	406.06	5.08

사례 1은 조건문으로 12가지 조건에 비교하며 마지막 조건에 참으로 구축하였다. 마지막 조건에 참으로 성립되면 줄 바꿈을 출력하고 조건문을 끝내게 된다. 조건문의 경우 총 7가지의 사례 중 성능 차이가 1.12배로 가장 차이가 적은 것을 알 수 있다.

사례 2는 버블정렬 알고리즘을 사용한 것이다. 최악의 상황을 주기 위하여 30부터 1까지의 역순으로 되어있는 배열 10개를 버블정렬로 정렬하기 위해 소요된 시간을 측정하였다.

사례 3은 정수형 변수를 통해 비교하는 반복하는 반복문이다. 비교를 통해 100회 실행이 되는 반복문이고 참일 때 함수를 호출한다. 호출한 함수는 문자열을 출력하는 함수이다. 사례 4는 사례 3과 동일한 정수형 변수 반복문을 사용한다. 사례 5는 사례 3, 4와 다르게 실수형 변수를 통해 비교하여 반복하는 반복문을 사용하였다. 자바스크립트에서는 변수형의 타입이 없어 let으로 선언하였고 각각 정수형과 실수형처럼 작동시키기 위해서 1과 1.0씩 더해지는 변수로 반복문을 진행하였다. 사례 6, 7도 사례 4와 같이 정수형 변수를 사용한 반복문이고 사례 4와 달리 반복문 횟수의 차이를 두었다. 사례 4, 6, 7 각각 100회, 10,000회, 100,000회의 횟수로 반복문을 사용하였다.

사례 4, 6, 7은 같은 반복문이지만 실행 횟수에 따라 자바스크립트와 웹 어셈블리 간의 차이가 벌어지는 것을 확인할 수 있다. 사례 3, 4는 같은 횟수의 반복문이지만 반복문 안에서 함수 호출과 덧셈을 한다는 내부적인 차이점으로 인해 웹 어셈블리에서 약 15us가 차이와 자바스크립트는 약 6164us인 웹 어셈블리의 약 410배 차이가 있다는 것을 확인할 수 있다. 또한, 사례 4, 5의 결과로 보았을 때 자바스크립트와 웹 어셈블리 모두 정수형과 실수형인 타입만으로도 시간에 차이가 있다는 것을 관찰할 수 있다.

3. 결과분석

측정 결과 조건문의 경우 자바스크립트가 웹 어셈블리보다 더 뛰어난 성능을 나타내는 것을 확인하였다. 반면에 반복문과 버블정렬은 자바스크립트보다 웹 어셈블리의 성능이 더 좋다는 것을 확인할 수 있었고 반복문의 경우 횟수가 커질수록 자바스크립트와 웹 어셈블리간의 실행 간격이 커지는 것을 확인할 수 있었다. 총 7가지의 사례를 통해 웹 어셈블리가 자바스크립트보다 성능이 더 뛰어나거나 비슷하다는 것을 관찰할 수 있었다.

Ⅲ. 결론

본 논문에서는 코드 단위별 웹 어셈블리와 자바스크립트의 성능에 대해서 실험해보았다. 하지만 이 실험은 단위 모듈 코드에 대한 성능 분석으로 복잡한 코드에서는 결과가 달라질 수 있다. 본 연구는 동일한 문맥에 해당하는 코드를 두 가지 각각 작성하고 그 코드에 대한 성능을 알아보았고 실험 결과 웹 어셈블리가 자바스크립트의 성능과 유사하거나 뛰어났다. 이런 현상으로 인해 웹 시장은 자바스크립트로 구현된 웹이 아닌 웹 어셈블리로 구현된 웹이 더 많아질 것이라고 예상된다. 또한, 자바스크립트의 역할이 기존의 웹을 구성하는 역할에서 웹 어셈블리가 웹에 실행될 수 있도록 도와주는 glue code의 역할만 하게 될 것으로 예측한다.

ACKNOWLEDGMENT

이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. 2022R1C1C1006967).

참 고 문 헌

[1] 신창현, 여지환, 문수묵, “웹어셈블리의 수행 과정 및 성능 분석” 한국컴퓨터종합학술대회 논문집 p. 1546-1548, 2018

[2] 송우석, 김덕형, 정현준, 정동원, “웹 어셈블리를 활용한 웹 환경의 Unity 기반 게임 성능 분석” 한국정보기술학회 추계 종합학술대회 논문집 p. 387-391, 2021

[3] 송우석, 정승원, 정현준, 정동원, “웹 어셈블리를 활요한 웹 환경의 블록체인 활용 가능성 분석” 한국정보기술학회 하계 종합학술대회 논문집 p. 255-259, 2022

[4] emscripten. 2015. emscripten. https://emscripten.org

[5] MDN Web Docs. 2022. WebAssembly. https://developer.mozilla.org/ko/docs/WebAssembly

[6] W3C. 2019. World Wide Web Consortium (W3C) brings a new language to the Web as WebAssembly becomes a W3C Recommendation. https://www.w3.org/2019/12/pressrelease-wasm-rec.html.en

[7] npm. 2017. nano-time. https://www.npmjs.com/package/nano-time