Xiang Li
xl68@rice.edu
November 29, 2016

WebAssembly: Future Web Platform?

Introduction

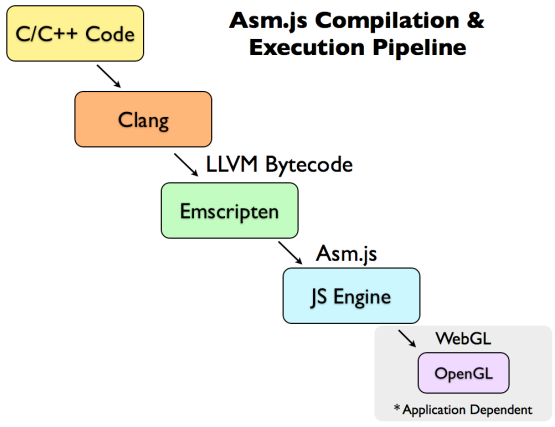
Since first announced on the 17th of June, 2015, WebAssembly ("wasm" for short), an ongoing project, has been viewed as the next-generation web platform widely in the web development community. Under the banner of the W3C Community Group, people from Google, Microsoft, Mozilla, Apple, and others have been developing WebAssembly since April 2015.

What exactly is WebAssembly and why does it seem so promising? The answers could be found in this article, together with my personal opinion on how WebAssembly will effect JavaScript, the currently dominant web development language.

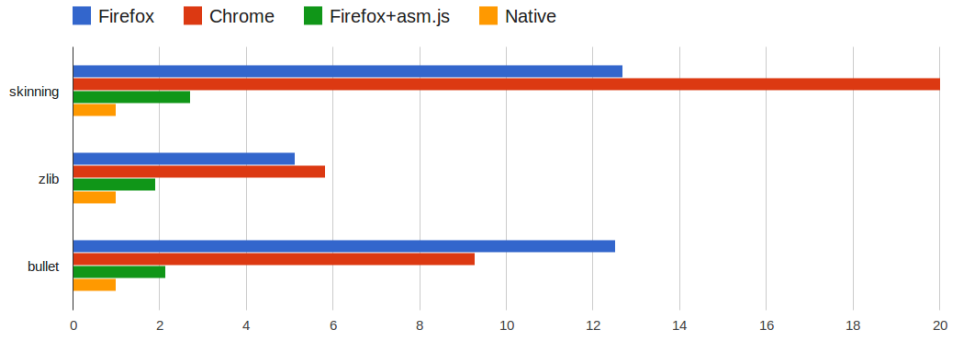
From asm.js to initial WebAssembly

Initially, WebAssembly is (roughly) a binary format for delivering asm.js code. Even though wasm now evolves to a different platform, it is a necessity to look back to the pioneering asm.js, which is a massive feat.

Developed by Mozilla in 2013-2014, asm.js is not a new programming language, but a strict and specialized subset of JavaScript. Actually, asm.js codes are JavaScript codes that only deal with numbers and nothing like normal DOM-using codes, so as to be converted into assembly directly. It comes from a new category of JavaScript application: C/C++ applications that have been compiled into JavaScript. With asm.js, and Emscripten, the compiler converts byte code generated by LLVM to asm.js, applications written



in C/C++ including game engines, graphics, and emulators could been run as web applications.



Run time normalized to Native (clang -O2), lower values are better

Some early experiments imply that the compiled complex C++ applications are only around 2 times slower than natively compiled applications, and 4 to 10 times faster than latest browser builds [1]. This result is very exciting since the improvement is so substantial, which indicates that asm.js is indeed very efficient. However, parsing JavaScript is still low, especially for mobile applications. Large compiled codes on mobile can easily take 20–40 seconds just to parse, which arises the call for fast native decoding.

WebAssembly, initially a binary format to deliver asm.js code, can be compiled to execute at native speed. Indeed, it can be natively decoded 20 times faster than JavaScript can be parsed [2]. Furthermore, WebAssembly could be evolved simpler, by avoiding the simultaneous asm.js constraints of Ahead-Of-Time Compilation within JavaScript syntax. Consequently, WebAssembly has a better performance than asm.js.

WebAssembly as a platform and its applications

C++	Binary	Text

`int factorial(int n) {`
`  if (n == 0)`
`  return 1;`
`  else`
`  return n * fac(n-1);`
`}`
	`20 00`
	`42 00`
	`51`
	`04 7e`
	`42 01`
	`05`
	`20 00`
	`20 00`
	`42 01`
	`7d`
	`10 00`
	`7e`
	`0b`
		`get_local 0`
		`i64.const 0`
		`i64.eq`
		`if i64`
		`  i64.const 1`
		`else`
		`  get_local 0`
		`  get_local 0`
		`  i64.const 1`
		`  i64.sub`
		`  call 0`
		`  i64.mul`
		`end`

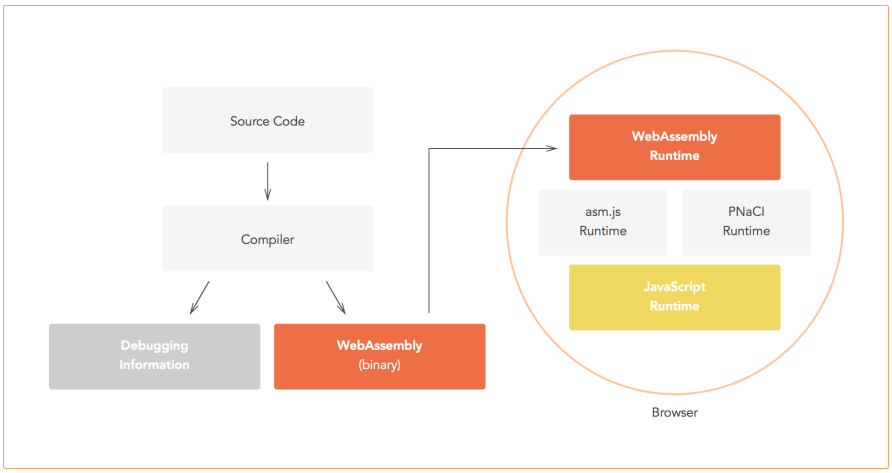
Facing the era of mobile and Internet of Things (IoT), as mentioned above, one of WebAssembly's high-level goals is to define such portable, size- and load-time-efficient binary format that can be compiled to execute at native speed.

WebAssembly binaries will encode abstract syntax trees. That is, they are not a linear format like stack- or register-based byte code. Not surprisingly, the format looks much like asm.js: roughly, a statically typed version of JavaScript without objects [3]. Since there are use cases including viewing source on a

WebAssembly module while debugging, or writing WebAssembly code directly for some reasons, a text format for WebAssembly is defined. As shown in the above figure, it will mirror the semantics of the binaries and contain a tree of statements and expressions [4].

In addition to developing Minimum Viable Product (MVP), which has roughly the same functionality as asm.js, WebAssembly is also adding new features. This includes some key features like threads, zero cost exceptions, and SIMD (single-instruction-multiple-data). More importantly, WebAssembly is trying to bring language diversity to the web platform. Unlike asm.js mainly for C/C++, multiple compilers could be developed in future to support different languages in web applications.

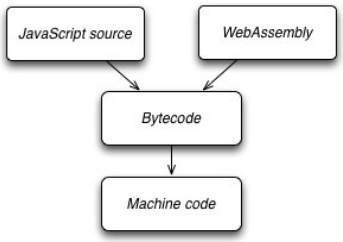
As you can see, WebAssembly is aimed at making a great next-generation web platform. The W3C WebAssembly Community Group is currently building a new LLVM backend for WebAssembly and an accompanying clang port, as well as collaborating with more compiler vendors for different languages in the future. Surely, wasm is designed to execute within and integrate well with the existing Web platform, and co-exist well with JavaScript. Specifically, wasm supports a JavaScript API for JavaScript to compile and manage WebAssembly modules, which are naturally integrated with the ES6 module system. Moreover, wasm is also designed to be able to execute well in other environments other than webs, including servers in data centers, on IoT devices, or mobile/desktop apps.



With these features added, WebAssembly would be used to develop more web applications. For instance, with threads and SIMD, wasm supports fat, parallel processing pipelines for realtime video stream effects processors. In fact, most common and obvious applications are in fields like games, virtual reality (VR), augmented reality (AR), image/music/video applications, scientific visualization and simulation, encryption, and so on. Moreover, there are also some applications most people don't think of when they think of JavaScript, such as apps that stream big channels of data through networks of processing functions.

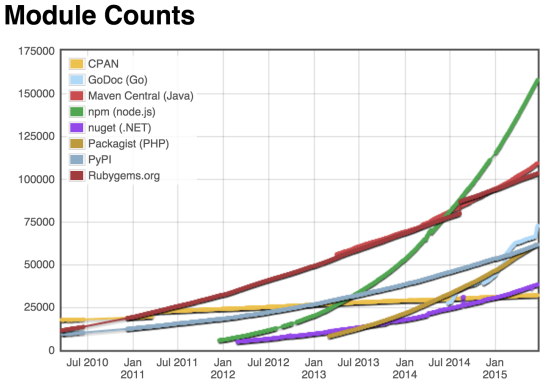
Besides applications inside the browser, since wasm is designed to support non-browser embeddings, it have applications outside the browser as well. Server-side application, game distribution service (portable and secure), hybrid native apps on mobile devices, or symmetric computations across multiple nodes are all possible applications [5].

Will WebAssembly replace Javascript?



Like the figure on the left shows , one may not help thinking whether wasm will replace Javascript in the future. Missing byte codes has been an issue for JavaScript for some time, but wasm now perfectly fill out the gap. If other languages could be coded to develop web applications well, JavaScript seems not very necessary to developers.

These are all reasonable concerns, yet as claimed by W3C WebAssembly Community Group, WebAssembly is not replacing, but helping JavaScript in the near future. Please recall that wasm's high-level goal is become a web platform, it does not have to replace JavaScript. Wasm is now designed to integrate with existing web platforms by supporting related APIs. Moreover, currently JavaScript is the dominant language and still rising in web development community, as shown in figure below. "We're not killing JavaScript. I don't think it's even possible to kill JavaScript", quoted from Brendan Eich (JavaScript founder, and member in W3C WebAssembly Community Group) [7].



Nevertheless, I think the reason why WebAssembly is supporting JavaScript is not it wants to, but it has to. To help wasm being accepted by the community and evolved to be better, it has to co-exist with current existing web platforms. But if there is a day when wasm really becomes a complete web platform, and every one is accustomed to develop web applications by other programming languages, why do we have to keep using JavaScript.

Conclusion

As we discussed, evolved from asm.js, WebAssembly is becoming increasingly promising as it is developed more completely as a web platform. By possibly compiling other languages into its binary format, running mainstream mobile/IoT applications/games in web (or on servers) is feasible and really interesting. In the near future, it will co-exist with JavaScript and integrate with existing web platforms well, though I personally believe it is possible that wasm would replace JavaScript one day.

Reference

1. http://ejohn.org/blog/asmjs-javascript-compile-target/
2. https://github.com/WebAssembly/design/blob/master/FAQ.md
3. http://www.2ality.com/2015/06/web-assembly.html
4. https://github.com/WebAssembly/design/blob/master/TextFormat.md
5. https://github.com/WebAssembly/design/blob/master/UseCases.md
6. https://medium.com/javascript-scene/what-is-webassembly-the-dawn-of-a-new-era-61256ec5a8f6#.odlkiqd3i
7. https://medium.com/javascript-scene/why-we-need-webassembly-an-interview-with-brendan-eich-7fb2a60b0723#.n77ff9p6c