From NaCl to WebAssembly

Exemplified by MainConcept® HEVC Decoder

AUTHOR: SERGEY BUFALOV, Manager Software Engineering at MainConcept

Executing native code inside web browsers has a long and rich history. This methodology is quite popular and common nowadays as it helps to optimize critical and resource intensive parts of web applications. For those developers looking for native code performance inside the secure environment of modern browsers, WebAssembly (WASM or WebASM) binary format has become a popular option. During the past few years, Mozilla, among others, has actively promoted this standard. Currently it is published by W3C (World Wide Web Consortium) as a draft for Release 1.0 [1].

MainConcept recognized the importance of web technologies for a wide range of multimedia applications and developed native codecs in the Google Native Client (NaCl) sandboxed environment. When it was deprecated in favor of the natively embedded WebAssembly binary format, MainConcept continued to release C/C++ codecs for the web and created the MainConcept WASM HEVC Decoder SDK product. The primary goal for the first version was to expose unmodified MainConcept callback APIs with inevitable speed regressions minimized. This goal has been achieved.

The base for efficient WASM code is the Emscripten C/C++ to JavaScript compiler. It makes the WASM HEVC Decoder creation task just a matter of compiling C/C++ code to another target. The generated bytecode is portable to any platform and any browser. To keep the first version as simple as possible, MainConcept elected not to use the feature testing framework of the WebAssembly standard. MainConcept also unconditionally disabled threads and SIMD in the source code. Such changes will soon no longer be required since browsers are actively adding support for missing WebAssembly features.

Even though the C/C++ codec to WASM can be compiled with minimum changes, it doesn't mean that the native performance is automatically available in a browser. The generated bytecode must be bound to JavaScript to become invocable from a web application, and the binding API must

not introduce overhead. In addition, the MainConcept development team required the API contain no hand-written JavaScript and be completely implemented in C/C++. The Emscripten compiler's support for WebIDL made it possible to bind WASM and JavaScript with a thin lightweight intermediate layer using IDL interface description and C/C++ interface implementation without any use of JavaScript.

The security requirements of modern browsers require JavaScript and WASM to use different heaps for memory allocations and prevent any direct access to each other's memory. Bytes stored in Uint8Array on the JavaScript side must be copied to a HEAPU8. buffer on the WASM side and vice versa. Emscripten's support of arrays in WebIDL allows us to hide heap management and keep the API simple and high performing, leaving the memory transfers from heap to heap the only serious overhead introduced by the binding layer.

While decoding an input stream, the MainConcept WASM HEVC Decoder continuously reports its state via dedicated callbacks defined by the IDL interface. Users must implement them on the JavaScript side and pass them to the C/C++ backend during decoder creation. Callback functions are invoked from inside native code using Emscripten's support of embedding JavaScript in C/C++. Function arguments are passed as class objects rather than raw pointers which makes it very convenient to use callbacks on the JavaScript side.

As expected, the WebAssembly standard specifies a very efficient binary format and has excellent support in the Emscripten compiler. However, it cannot preserve the level of effectiveness reachable in C/C++. The table below illustrates the speed degradation obtained on an Intel® Core™ i7-8700K @ 3.7GHz processor with 32GB RAM. The difference between columns "WASM Codec" and "SIMD Disabled" illustrates the overhead introduced by WASM binary format, WASM heap management, and JavaScript bindings. As one can see, the overhead is about 1/3 of the native codec with SIMD and threads disabled.

Stream	FPS			
	Native Codec	Threads Disabled	SIMD Disabled	WASM Codec
720p 2 Mbit/s Main 4:2:0	1406.5	282.9	91.3	62.2
1080p 4 Mbit/s Main 4:2:0	768.5	127.0	38.7	26.5
2160p 6 Mbit/s Main 4:2:0	282.0	45.4	11.6	7.7

Table 1. The evolution of speed degradation from native HEVC Decoder to WASM Decoder.

The MainConcept WASM HEVC Decoder provides the complete set of C/C++ HEVC Decoder features (up to 14-bit 4:4:4 8K) inside all browsers supporting WebAssembly. Keeping the code base completely in the C/C++ domain guarantees the performance and quality based on a wide set of existing C/C++ optimization and verification techniques and methodologies. MainConcept is proud to make this innovative new product available to our customers to demo or buy today.

[1] https://webassembly.github.io/spec

REQUEST A DEMO:

MAINCONCEPT
WEBASM HEVC DECODER

Free evaluation downloads are available for testing.

CONTACT

MainConcept GmbH
Elisabethstr. 1, 52062 Aachen
GERMANY

marketing@mainconcept.com
www.mainconcept.com/webasm

AUTHOR

SERGEY BUFALOV
Manager Software Engineering
at MainConcept