Web Audio API vs. native, closing the gap, take 2

[Extended Abstract]

Paul Adenot
Mozilla Corporation
padenot@mozilla.com

1. TOPICS

My keynote at the first Web Audio Conference at IRCAM in Paris attempted to answer this question, but it was clearly too early in the lifetime of the Web Audio API to have a definitive answer.

The situation has evolved quite a lot, with the addition of various features, not necessarily related to audio.

Shared memory, atomics, SIMD, Audio Worklets, WASM, Web MIDI are some of the features that allow building programs that were impossible to write a few years ago.

This workshop will take a bottom up approach to writing high-performance applications with the Web Audio API, with a definitive focus towards writing real-time audio code in the context of a web application.

In doing this, the reasoning will follow a bottom-up approach: understanding the properties needed for a specific system, and try to them map web platform constructs, with a definitive focus on high-performance and white-box analysis. Links to implementations themselves and the primitives chosen will have an impact on the final quality of the result, in terms of rendering speed, memory footprint, robustness, and extensibility.

The limitations of the web platform will be discussed, with possible workarounds, along with the multiple efforts are in the works to remove those limitations.

1.1 Audio Worklets

`AudioWorklet` is the back-bone of the real-time audio processing on the web, outside of the pre-defined `AudioNode` that have been available for a long time. It provides a way to execute script as part of the rendering of the audio, with a corresponding main thread object that has custom `AudioParam` exposed, along with a `MessagePort`.

The rendering thread side has the other end of the `MessagePort`, and must have a method called `process` with a number of parameters, where all the signal computations happen. This method received the input of the node (if applicable), has a buffer where the output signal can be written, and has a third buffer with the value of the `AudioParam`.

1.2 WASM

WASM (Web Assembly) plays a very important role in moving towards native-like performances, because it allows using a language that does not use a garbage collector. Because the audio latencies on modern platforms are very low, blocking the audio rendering thread, even what seems to be short periods of time, is problematic.

Additionally, it executes most of the time much faster than JavaScript, and allows authoring the signal processing algorithm in languages that are better suited to the task at hand (C++, Rust, Faust, etc.).

1.3 Shared memory

Shared memory is now available on the web, with the object `SharedArrayBuffer`. It can be transfered via a `MessagePort`, and then its content can be mutated racily on multiple threads. Shared memory is a powerful tool, but authors should have a certain discipline when using it. Special care must be taken when using it, from a software engineering but also performance standpoint.

This construct has quite a few security implications, that will be quickly discussed, and good practices for deployment of programs using shared memory will be provided.

1.4 Web Workers

Web Worker opens the door to multi-thread computing and concurrency. This construct has been available for a long time on the web platform, but it has been made more powerful recently thanks to the availability of shared memory.

This construct will be looked at from an operating system perspective, to understand in which circumstances it is possible and feasible to offload some processing to a Web Worker, and why.

1.5 Atomics

Atomics, along with 'SharedArrayBuffer', allow implementing lock and wait-free algorithms on the web platform, that are essential for audio. In particular, the wait-free single-producer single-consumer ring-buffer (the bread and butter of audio programming), will be investigated.

Web Audio Conference WAC-2019, December 4–6, 2019, Trondheim, Norway.

1.6 SIMD

SIMD (Single Instruction Multiple Data) is a way to optimize computation, on a single thread, via operating on multiple scalar values at once. Often, algorithms need to be tweaked to make efficient use of SIMD. A couple approaches useful to deal with the specifics of audio algorithm will be explained.

1.7 Web MIDI

Web MIDI is essential to access external hardware: control surfaces, keyboard for input, analog synthesizers and other drum machine on the output. Special care must be taken when integrating Web MIDI and the Web Audio API, in particular when it comes to timing.

2. SYNOPSIS

A straightforward signal processing algorithm has been chosen for illustration purposes, and a number of signal processing primitives are provided in languages that are suited to the task at hand. The first task is to have a working effect, by implementing the glue code, compiling the result to WASM, and to write the JavaScript code to load the WASM module in an `AudioWorklet`.

The effect is now working but no controls are provided. Control of the effect via either `AudioParam`s or shared memory and atomics is implemented, along with potential optimizations via `SIMD`, and performance measurements performed. A discussion starts on what contributes to high-performance algorithms and on how to measure performance of a real-time algorithm on the web platform and in general, based on the findings of the group.

Control via Web MIDI or other means (OSC using WebSockets or WebRTC) to integrate with other systems will be implemented. Packaging into a standard format (Web Audio Modules) is discussed.

With a sort of gap analysis, the participants that have prior knowledge developing native audio processing algorithms are reflect on what the web platform provides, and what is missing. Possible solutions are explored, with pointers to existing web platform efforts (discussions, events, issues opened on specifications, other specifications, etc.).