



FFmpeg and WebAssembly

WA

Aditya Sharma

Project: Porting FFmpeg to WebAssembly

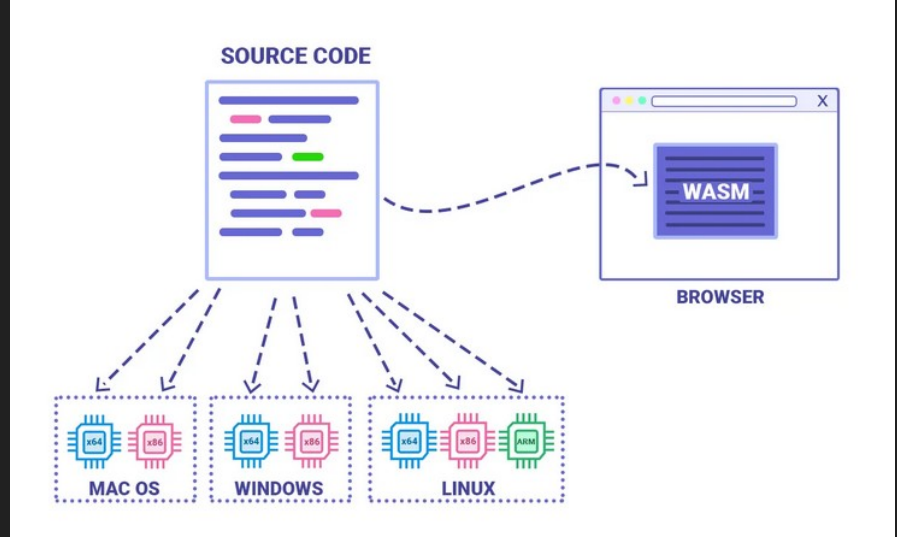
Timeline and milestones:

1. Understand WebAssembly and Emscripten as tool to port native C/C++ applications to WebAssembly
2. Port the open source project FFmpeg along with its libraries and tools like ffprobe and ffplay using Emscripten.
3. Run the ported libraries in web browser and implement the transcoding feature using FFmpeg WebAssembly functions.
4. Benchmarking the ported application with native tools and profiling the heap memory and performance parameters of the application.
5. (Extension of analysis) Explore various flags and options for optimizing the performance.
6. Communication with open source community to overcome roadblocks.

Significance:

Source: <https://blog.logrocket.com/webassembly-how-and-why-559b7f96cd71/>

- Javascript not designed for fast computations. It is designed to provide interaction to the web page.
- Wasm can provide bring the level of efficiency that native c/c++ have.
- Wasm + javascript combination essentially means that the application will run in browser irrespective of the Operating System hosting the browser.



- “Wasm is usually slower by about 10% than native C code.” -

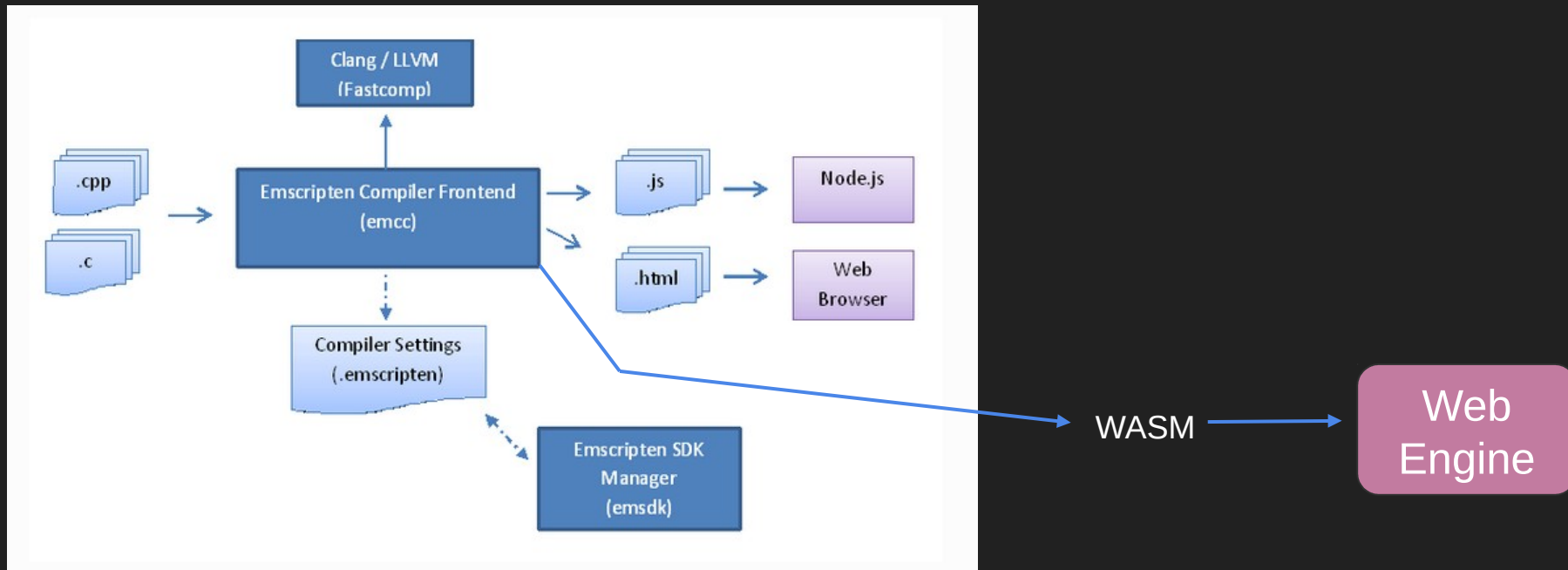
<https://medium.com/@torch2424/webassembly-is-fast-a-real-world-benchmark-of-webassembly-vs-es6-d85a23f8e193>

<https://www.usenix.org/system/files/atc19-jangda.pdf> 2019 Paper.

- There is a high commercial utility of FFmpeg port. FFmpeg is a very popular open source library therefore it is clear that web port will also have commercial utility.

Workflow and Implementation:

1. WebAssembly Application



Source: https://emscripten.org/docs/introducing_emscripten/about_emscripten.html

Emscripten compiler (emcc) uses Clang and LLVM to compile c/c++ to wasm or asm.js. The javascript emitted provides necessary runtime support and helps running the compiled code.

Emscripten support is comprehensive. It supports C/C++ standard libraries. It also has support for other libraries like pthreads, SDL, OpenGL, zlib and many more. In short, correctness is guaranteed for almost all the cases. However, sometimes it is necessary to make small changes to ensure that the port works. This will be revisited later in the presentation.

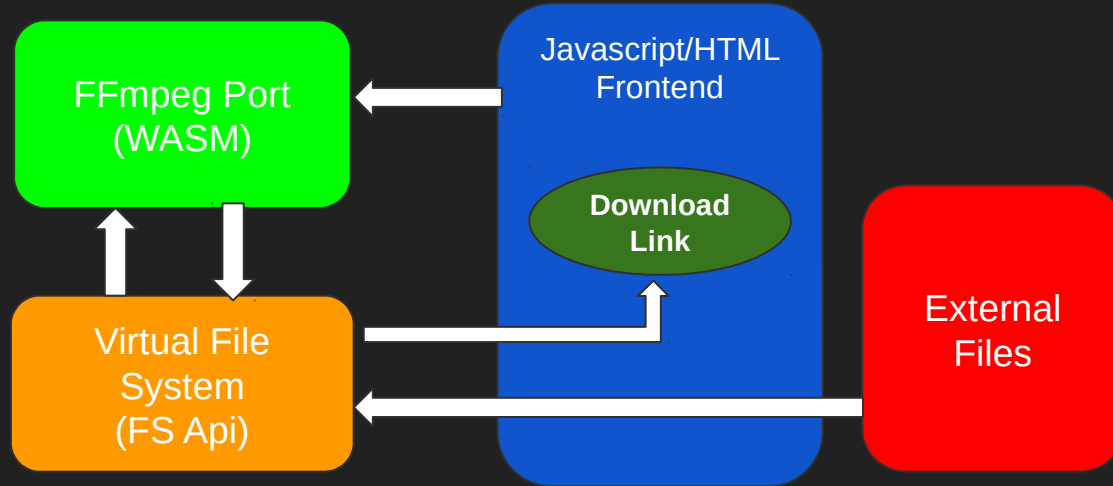
C++, C or Rust



Source: <https://blog.logrocket.com/webassembly-how-and-why-559b7f96cd71/>

Workflow and Implementation:

2. FFmpeg Ported application



Workflow:

1. The Javascript/HTML front end requests file for transcoding.
2. FS api writes the file in Virtual File system that is created for FFmpeg port.
3. Script prepares a request for ffmpeg port that sends all relevant information for the process.
4. FFmpeg directly interacts with the virtual filesystem. It stores the processed output within the same filesystem.
5. Script loads the file and creates a downloadable Blob object along with a link to download it to the local file system.

Porting Procedure: A brief overview

1. Install and setup Emscripten from their official github repo and follow the instruction on the official website.
2. Download the latest stable version from <https://www.ffmpeg.org/download.html>. This will be used as our native library for porting.
3. Use emscripten to configure and build library.(emconfigure, emmake and emcc are counterparts of configure, make and gcc).

```
emconfigure ./configure --disable-x86asm --disable-inline-asm --disable-doc --disable-stripping --nm="llvm-nm -g" --ar=emar --cc=emcc --cxx=em++ --objcc=emcc --dep-cc=emcc
```

```
emcc -lLibavcodec -lLibavdevice -lLibavfilter -lLibavformat -lLibavresample -lLibavutil -lLibpostproc -lLibswscale -lLibswresample -Qunused-arguments -Oz -o FfmpegJavascript/ffmpeg-script.js fftools/ffmpeg_opt.o fftools/ffmpeg_filter.o fftools/ffmpeg_hw.o fftools/cmdutils.o fftools/ffmpeg.o -lavdevice -lavfilter -lavformat -lavcodec -lswresample -lswscale -lavutil -lm -s EXPORTED_FUNCTIONS="[_ffmpeg]" -s EXTRA_EXPORTED_RUNTIME_METHODS="[cwrap, FS, getValue, setValue]" -s TOTAL_MEMORY=33554432 -s ALLOW_MEMORY_GROWTH=1 -s USE_SDL=2
```

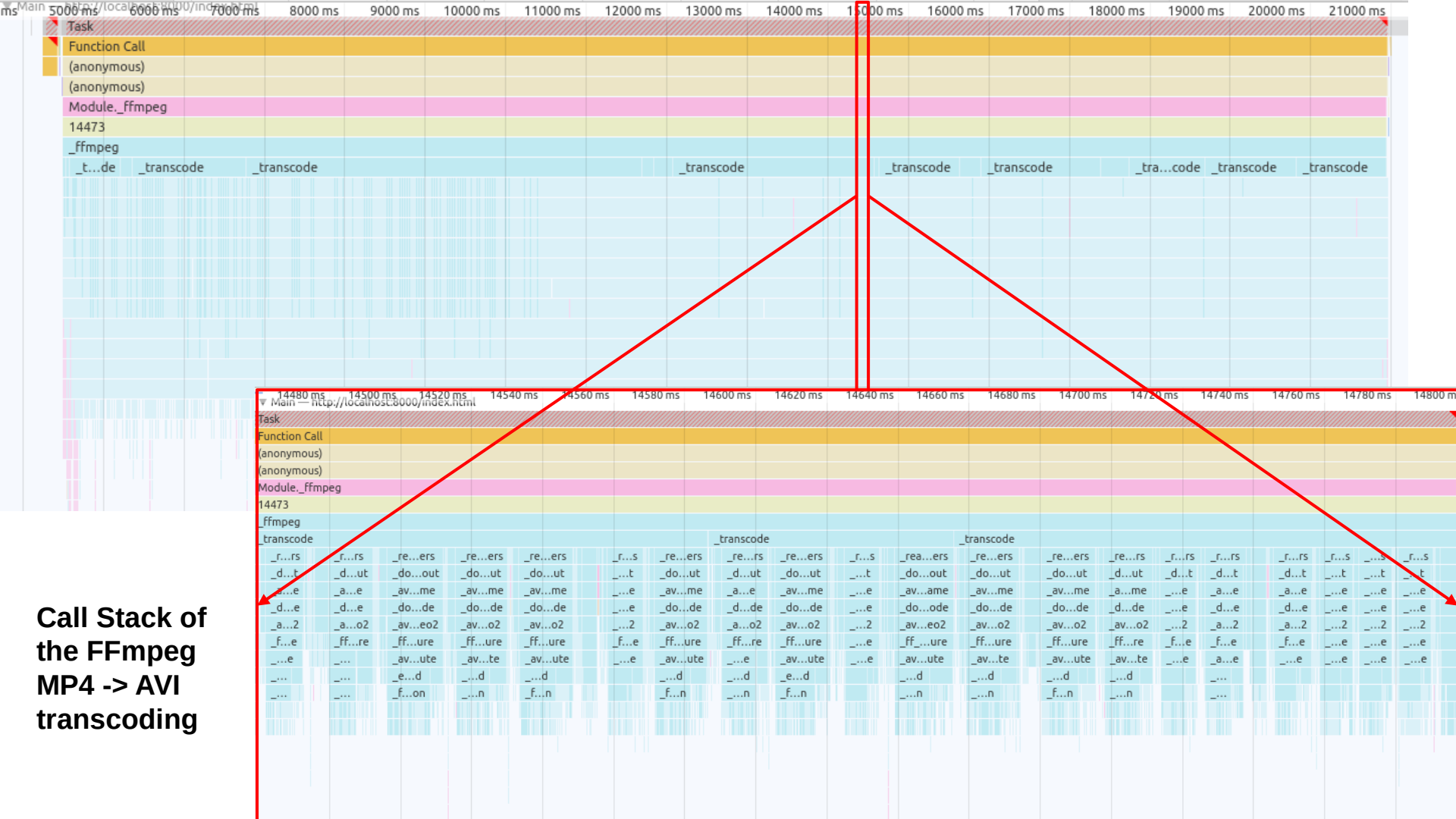


```
emconfigure ./configure --disable-x86asm --disable-inline-asm --disable-doc --disable-  
stripping --nm="llvm-nm -g" --ar=emar --cc=emcc --cxx=em++ --objcc=emcc --dep-cc=emcc  
--disable-ffmpeg --disable-ffplay
```

```
emcc -Llibavcodec -Llibavdevice -Llibavfilter -Llibavformat -Llibavresample -Llibavutil  
-Llibpostproc -Llibswscale -Llibswresample -o FFprobeJavascript/ffprobe-script.js  
fftools/cmdutils.o fftools/ffprobe.o -lavdevice -lavfilter -lavformat -lavcodec  
-lswresample -lswscale -lavutil -lm -s USE_SDL=2 -s EXPORTED_FUNCTIONS="[_ffprobe]" -s  
EXTRA_EXPORTED_RUNTIME_METHODS="[cwrap, FS, getValue, setValue]" -s TOTAL_MEMORY=33554432  
-s ALLOW_MEMORY_GROWTH=1
```

4. Develop frontend compatible with the js and wasm files ported by emscripten.
5. Use a local server to host your application and test transcoding and various features of the ported library.

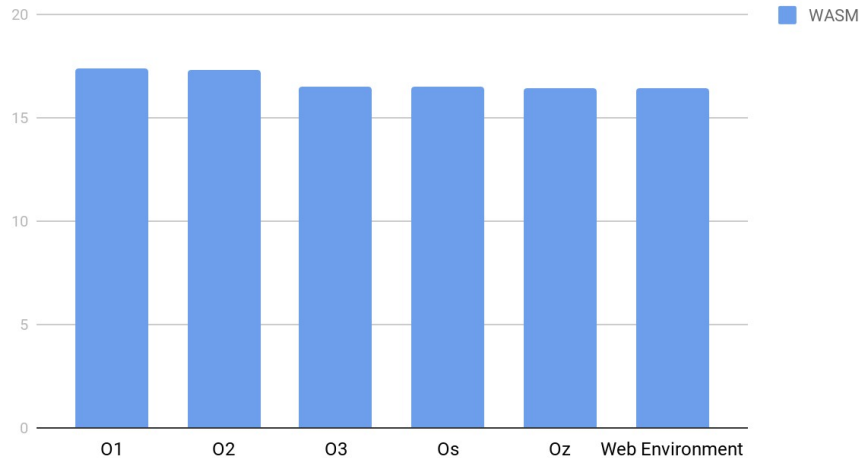
Note that the detailed process is documented as a reference file that is available on github repository.



Benchmarking and Performance

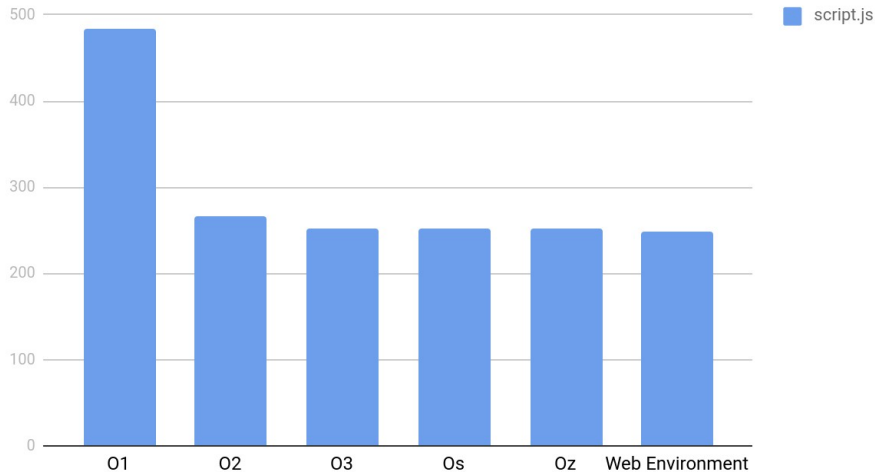
Build Sizes from Optimization flags

File Sizes



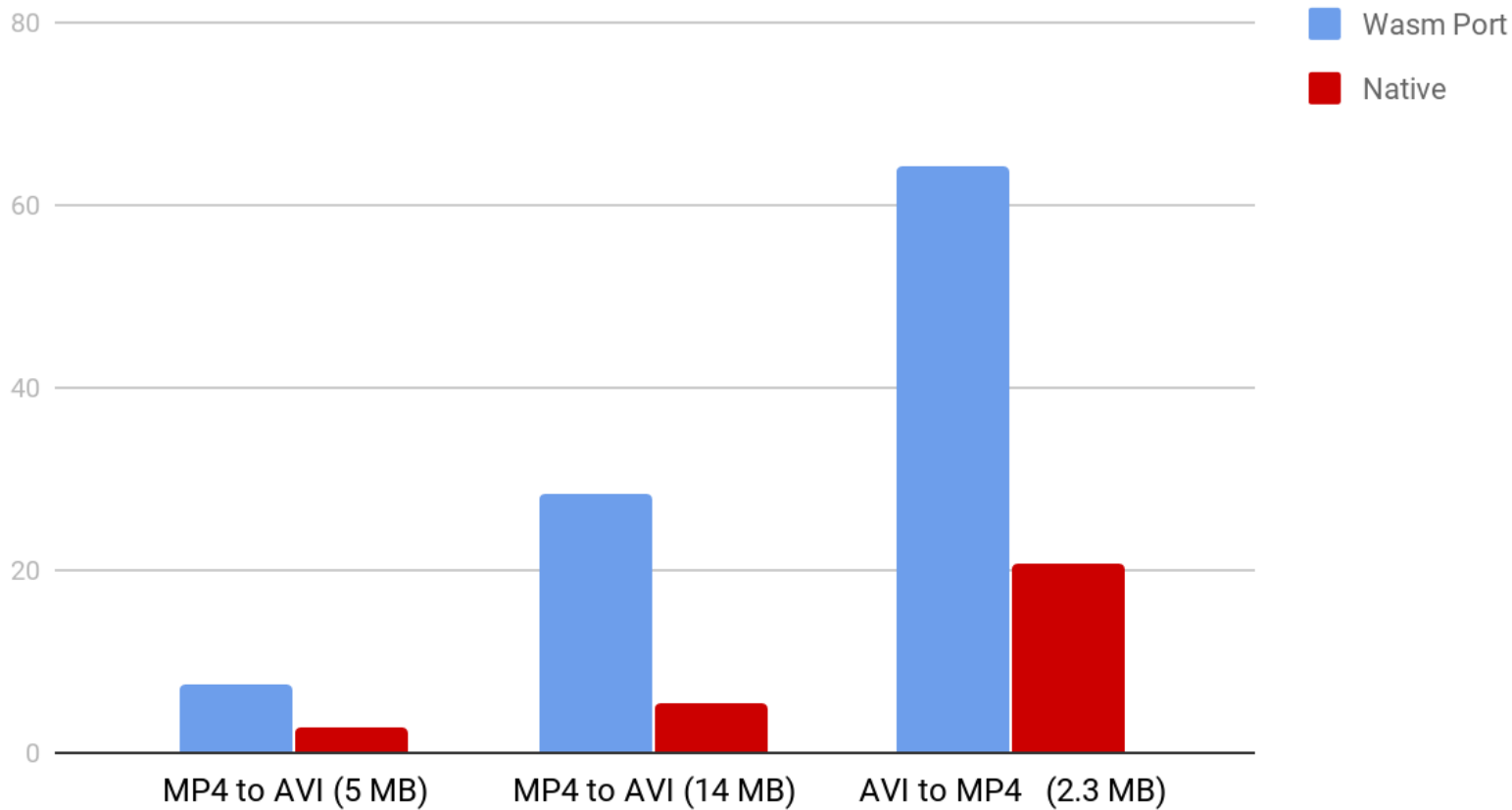
Size in Megabytes

File Sizes



Size in Kilobytes

Execution Time



Key Observations:

1. The performance of native is 2-3 times faster than the ported application. The performance ratio becomes better when longer codes are executed.
 - Native code has the advantage of less overhead as compared to browser processes. Also User Time + System Time is used as the counterpart metric for benchmarking Native code. While Devtools are used for WASM. May not be a very apt comparison.
 - This can be improved(discussed later)
2. Optimization flags have significant effect on the size of the wasm and js files.

18737.2 ms	23.15 %	18737.2 ms	23.15 %	▼_pix_abs16_c
15562.3 ms	19.22 %	15562.3 ms	19.22 %	▼_ff_epzs_motion_search
15562.3 ms	19.22 %	15562.3 ms	19.22 %	▼_ff_estimate_p_frame_motion
15562.3 ms	19.22 %	15562.3 ms	19.22 %	▼_estimate_motion_thread
15562.3 ms	19.22 %	15562.3 ms	19.22 %	▼_avcodec_default_execute
15562.3 ms	19.22 %	15562.3 ms	19.22 %	▼_ff_mpv_encode_picture
15562.3 ms	19.22 %	15562.3 ms	19.22 %	▶_avcodec_encode_video2
3157.6 ms	3.90 %	3157.6 ms	3.90 %	▶_ff_estimate_p_frame_motion
17.3 ms	0.02 %	17.3 ms	0.02 %	▶_estimate_motion_thread
9007.5 ms	11.13 %	9007.5 ms	11.13 %	▶_pix_abs16_xy2_c
4401.1 ms	5.44 %	4401.1 ms	5.44 %	▶_pix_abs8_c
4234.2 ms	5.23 %	4234.2 ms	5.23 %	▶_pix_abs16_y2_c
3823.5 ms	4.72 %	3823.5 ms	4.72 %	▶_pix_abs16_x2_c
2551.4 ms	3.15 %	2551.4 ms	3.15 %	▶_diff_pixels_c
2383.1 ms	2.94 %	2383.1 ms	2.94 %	▶_sse16_c
2131.6 ms	2.63 %	2131.6 ms	2.63 %	▶_ff_jpeg_fdct_islow_8
2072.5 ms	2.56 %	17234.9 ms	21.29 %	▼_encode_thread
2072.2 ms	2.56 %	17234.5 ms	21.29 %	▼_avcodec_default_execute
2072.2 ms	2.56 %	17234.5 ms	21.29 %	▼_ff_mpv_encode_picture
2072.2 ms	2.56 %	17234.5 ms	21.29 %	▶_avcodec_encode_video2
0.3 ms	0.00 %	0.3 ms	0.00 %	▶_ff_mpv_encode_picture
1991.9 ms	2.46 %	1991.9 ms	2.46 %	▶_pix_norm1_c
1915.6 ms	2.37 %	17478.2 ms	21.59 %	▶_ff_epzs_motion_search
1828.6 ms	2.26 %	45717.2 ms	56.47 %	▼_ff_estimate_p_frame_motion
1820.2 ms	2.25 %	45708.8 ms	56.46 %	▼_estimate_motion_thread
1820.2 ms	2.25 %	45708.8 ms	56.46 %	▼_avcodec_default_execute
1820.2 ms	2.25 %	45708.8 ms	56.46 %	▼_ff_mpv_encode_picture
1820.2 ms	2.25 %	45708.8 ms	56.46 %	▶_avcodec_encode_video2

Javascript profiler clearly shows that `_avcodec_encode_video2` consumes maximum time in a process

Heap Snapshot:

MP4->AVI (File Size 3MB)

▶ Window / http://localhost:8000	1	36	0 %	42 796 040	98 %
▶ ArrayBuffer ×11	2	532	0 %	41 789 692	95 %
▶ system / JSArrayBufferData ×7	3	41 788 548	95 %	41 788 548	95 %
▶ Uint8Array ×5	2	340	0 %	8 232 616	19 %
▶ Object ×229	2	7 668	0 %	5 340 278	12 %
▶ FSNode ×22	3	1 320	0 %	5 121 298	12 %
▶ (system) ×40529	2	875 148	2 %	1 074 456	2 %
▶ (array) ×856	2	495 952	1 %	822 944	2 %
▶ (compiled code) ×3770	3	384 136	1 %	527 892	1 %
▶ Table	2	32	0 %	243 972	1 %
▶ (closure) ×3462	2	101 724	0 %	236 704	1 %
▶ Window /	1	36	0 %	180 660	0 %
▶ (string) ×5396	2	145 708	0 %	145 708	0 %
▶ Instance	4	204	0 %	125 056	0 %
▶ Window ×9	2	232	0 %	70 992	0 %
▶ Object /	1	20	0 %	70 288	0 %
▶ Array ×85	2	1 360	0 %	66 528	0 %
▶ Error ×28	3	776	0 %	59 484	0 %
▶ (concatenated string) ×663	4	13 260	0 %	24 268	0 %
▶ Float32Array ×257	2	17 476	0 %	17 476	0 %
▶ Int32Array ×257	2	17 476	0 %	17 476	0 %
▶ Document ×2	4	40	0 %	16 824	0 %
▶ HTMLInputElement ×5	3	104	0 %	12 424	0 %

HEAP SNAPSHOTS



Snapshot 1
35.4 MB



Snapshot 2
42.2 MB



Snapshot 3
42.5 MB



Snapshot 4
43.0 MB

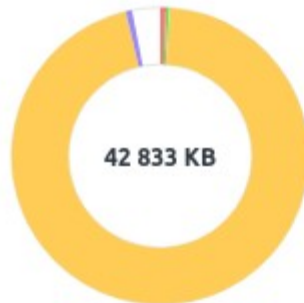


Snapshot 5
43.2 MB



Snapshot 6
43.9 MB

[Save](#)



274 KB Code
134 KB Strings
23 KB JS Arrays
40 810 KB Typed Arrays
286 KB System Objects
42 833 KB Total

Comparison with Native Run

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
1712	adityasca	20	0	246M	29288	13940	S	171.	0.4	0:02.07	./ffmpeg -nostdin -t

Native run requires about 29288 KB or
28.6MB

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
1887	adityasca	20	0	25.0G	291M	82872	R	103.	3.7	1:59.83	/opt/google/chrome/c

But the total memory footprint of the
browser process running transcoding
is around 291 MB

Shortcomings and how to overcome them:

1. The efficiency is still not upto the mark that can be achieved through web assembly.
 - a. Bringing GPU into picture: GPU support can decrease the time of execution and help increase efficiency.
 - b. Platform Specific Host Binding: Using platform specific bindings instead of generic APIs.
 - c. Cutting down debugging info: Current build involves lot of debugging info. We can reduce this to increase efficiency.
2. Porting is done using emscripten fastcomp backend. This is not a limitation as such, but it can be extended to llvm upstream with some modifications.
 - a. Can be done given some time.

Challenges faced and steps taken

1. Choosing configuration settings and compile flags. (FFmpeg)

```
emconfigure ./configure --disable-x86asm --disable-inline-asm --disable-doc --disable-  
stripping --nm="llym-nm -g" --ar=emar --cc=emcc --cxx=em++ --objcc=emcc --dep-cc=emcc
```

Emscripten does not
support assembly
functions.
To allow memory to
grow as needed

Replace compilers with
emscripten versions

Exported function
we will be using


Increasing
memory for
initial compile

Functions exported for our use

```
emcc -Llibavcodec -Llibavdevice -Llibavfilter -Llibavformat -Llibavresample -Llibavutil  
-Llibpostproc -Llibswscale -Llibswresample -Qunused-arguments -Oz -o  
FFmpegJavascript/ffmpeg-script.js fftools/ffmpeg_opt.o fftools/ffmpeg_filter.o  
fftools/ffmpeg_hw.o fftools/cmdutils.o fftools/ffmpeg.o -lavdevice -lavfilter -lavformat  
-lavcodec -lswresample -lswscale -lavutil -lm -s EXPORTED_FUNCTIONS="[_ffmpeg]" -s  
EXTRA_EXPORTED_RUNTIME_METHODS="[cwrap, FS, getValue, setValue]" -s TOTAL_MEMORY=33554432  
-s ALLOW_MEMORY_GROWTH=1 -s USE_SDL=2
```

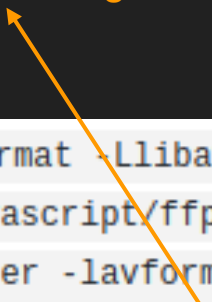
1. Choosing configuration settings and compile flags. (FFprobe)

```
emconfigure ./configure --disable-x86asm --disable-inline-asm --disable-doc --disable-  
stripping --nm="llvm-nm -g" --ar=emar --cc=emcc --cxx=em++ --objcc=emcc  
--disable-ffmpeg --disable-ffplay
```



Disable ffmpeg and ffprobe

Exported function
we will be using



```
emcc -Llibavcodec -Llibavdevice -Llibavfilter -Llibavformat -Llibavresample -Llibavutil  
-Llibpostproc -Llibswscale -Llibswresample -o FFprobeJavascript/ffprobe-script.js  
fftools/cmdutils.o fftools/ffprobe.o -lavdevice -lavfilter -lavformat -lavcodec  
-lswresample -lswscale -lavutil -lm -s USE_SDL=2 -s EXPORTED_FUNCTIONS="[_ffprobe]" -s  
EXTRA_EXPORTED_RUNTIME_METHODS="[cwrap, FS, getValue, setValue]" -s TOTAL_MEMORY=33554432  
-s ALLOW_MEMORY_GROWTH=1
```

2. How can FFmpeg port interact with files? FFmpeg in native form runs on the linux file system and reads/writes files directly. But how can a sandboxed browser provide the same interface?

Solution: FS API that is supported by Emscripten itself.

▼ 2 :: Window / localhost:8000 @4343	1	36	0 %	2 415 730	3 %
▶ buffer :: ArrayBuffer @119523	2	56	0 %	69 730 360	86 %
▶ FS :: Object @31451	2	504	0 %	7 192 641	9 %
▶ data :: Uint8Array @119521	2	68	0 %	2 279 918	3 %
▶ wasmTable :: Table @31551	2	32	0 %	243 972	0 %

FS uses the heap storage and creates a virtual File System that provides an interface for FFmpeg

First we write input file to FS. FFmpeg reads from FS and then writes back modified file to FS. Javascript reads and prepares an object that can be downloaded.

3. FFprobe port had the parser broken. Code exits with a runtime error on a virtual method being called on NULL value.

06082009.mp4

[index.html:56](#)

Invalid function pointer 18 called with signature 'iiii'. Perhaps this is an invalid value (e.g. caused [ffprobe-script.js:1751](#) by calling a virtual method on a NULL pointer)? Or calling a function with an incorrect type, which will fail? (it is worth building your source files with -Werror (warnings are errors), as warnings can indicate undefined behavior which can cause this). Build with ASSERTIONS=2 for more info.

⚠ ▶ Invalid function pointer 18 called with signature 'iiii'. Perhaps this is an invalid value (e.g. [ffprobe-script.js:1752](#) caused by calling a virtual method on a NULL pointer)? Or calling a function with an incorrect type, which will fail? (it is worth building your source files with -Werror (warnings are errors), as warnings can indicate undefined behavior which can cause this). Build with ASSERTIONS=2 for more info.

✖ ▶ Uncaught (in promise) RuntimeError: abort(Invalid function pointer 18 called with signature 'iiii'. [ffprobe-script.js:1763](#) Perhaps this is an invalid value (e.g. caused by calling a virtual method on a NULL pointer)? Or calling a function with an incorrect type, which will fail? (it is worth building your source files with -Werror (warnings are errors), as warnings can indicate undefined behavior which can cause this). Build with ASSERTIONS=2 for more info.) at Error

```
at jsStackTrace (http://localhost:8000/ffprobe-script.js:2012:17)
at stackTrace (http://localhost:8000/ffprobe-script.js:2029:16)
at abort (http://localhost:8000/ffprobe-script.js:1757:44)
at abortFnPtrError (http://localhost:8000/ffprobe-script.js:1491:2)
at nullFunc iiii (http://localhost:8000/ffprobe-script.js:8947:29)
at b18 (wasm-function[14281]:0x1bde3ef)
at _write_option (wasm-function[333]:0x222e91)
at _parse_option (wasm-function[332]:0x222776)
at _parse_options (wasm-function[334]:0x2230ad)
at _ffprobe (wasm-function[378]:0x22a954)
at abort (http://localhost:8000/ffprobe-script.js:1763:9)
at abortFnPtrError (http://localhost:8000/ffprobe-script.js:1491:2)
at nullFunc iiii (http://localhost:8000/ffprobe-script.js:8947:29)
at b18 (wasm-function[14281]:0x1bde3ef)
at _write_option (wasm-function[333]:0x222e91)
at _parse_option (wasm-function[332]:0x222776)
at _parse_options (wasm-function[334]:0x2230ad)
at _ffprobe (wasm-function[378]:0x22a954)
at Module._ffprobe (http://localhost:8000/ffprobe-script.js:9126:36)
at ccall (http://localhost:8000/ffprobe-script.js:820:18)
```

Issue with FFprobe
parsing arguments
given to the wasm
function

```
int main(int argc, char **argv)
{
    const Writer *w;
    WriterContext *wctx;
    char *buf;
    char *w_name = NULL, *w_args = NULL;
    int ret, i;

    init_dynload();

#if HAVE_THREADS
    ret = pthread_mutex_init(&log_mutex, NULL);
    if (ret != 0) {
        goto end;
    }
#endif

    av_log_set_flags(AV_LOG_SKIP_REPEATED);
    register_exit(ffprobe_cleanup);

    options = real_options;
    parse_loglevel(argc, argv, options);
    avformat_network_init();
    init_opts();
#if CONFIG_AVDEVICE
    avdevice_register_all();
#endif

    show_banner(argc, argv, options);
    parse_options(NULL, argc, argv, options, opt_input_file);
}
```

Notice that a NULL value has been passed to parse_options.

If NULL value has to be passed, why use this as an argument at all?


```

void parse_options(void *optctx, int argc, char **argv, const OptionDef *options,
void (*parse_arg_function)(void *, const char*))
{
    if ((ret = parse_option(optctx, opt, argv[optindex], options)) < 0)
        exit_program(1);
    optindex += ret;
}

```

```

int parse_option(void *optctx, const char *opt, const char *arg,
const OptionDef *options)
{
    const OptionDef *po;
    int ret;

    ret = write_option(optctx, po, opt, arg);
    if (ret < 0)
        return ret;
}

```

Same NULL value passed through all the functions. Possible access of this value resulted in the error.

FIX: Modify the parser functions to remove this value and comment all access points of null values. Need to disable ffmpeg for ffmpeg probe built.

```
static int write_option(const OptionDef *po, const char *opt,  
                        const char *arg)  
{  
    /* new-style options contain an offset into optctx, old-style address of  
    * a global var*/  
    // void *dst = po->flags & (OPT_OFFSET | OPT_SPEC) ?  
    //             (uint8_t *)optctx + po->u.off : po->u.dst_ptr;  
    void *dst = po->u.dst_ptr;
```

As mentioned, change parser and helper functions and comment lines where the variable optctx is referenced.

Scope for Improvement and Future:

1. This can be used as a starting point for targets with WebAssembly support.
2. Extending port to use exploit hardware accelerators that can significantly improve performance.
3. Reducing code size for better load performance and less load delay.
 - a. Can be done manually
 - b. Can be done using compiler optimizations like disabling catching errors, etc
4. Submit it to <https://github.com/emscripten-ports> as a library.

My Learning Experience:

1. Learnt about WebAssembly web technology(working, building and implementation)
2. Got to know basic architecture of Chromium(execution environment) and Browser threads.
3. Used software tools like Emscripten, Chrome DevTools.
4. Learnt new models and principles like Web-Worker model and how javascript actually implements this.
5. Developed a sense of interaction and collaboration with open source community and my mentor.
 - First time I actually posted an issue on Github and responded positively to comments and solutions
6. Realised the utility(commercial and technical) open source libraries have and how it can be extended
7. Technical aspects of software development like Benchmarking and Profiling.

Thank You