[GSoC] OpenCV.js: WASM SIMD optimization 2.0

KunLiang | [github](https://github.com/lionkunonly) | [gmail](lionkun1223@gmail.com)

Overview

* Proposal: [OpenCV.js: WASM SIMD optimization 2.0](https://docs.google.com/document/d/1a59OL-gmlyywRYw2Dui07u_DvsuqqlL9GuBr52hA9YU/edit?usp=sharing)
* Mentor: Vitaly Tuzov, Ningxin Hu
* Organization: OpenCV

Introduction

OpenCV.js is a JavaScript binding for selected subset of OpenCV functions for the web platform. It allows emerging web applications with multimedia processing to benefit from the wide variety of vision functions available in OpenCV. OpenCV.js leverages Emscripten to compile OpenCV functions into asm.js or WebAssembly targets, and provides a JavaScript APIs for web application to access them. In the last year, the SIMD and threads optimization had been applied on the OpenCV.js, which brought significant progress.

Though SIMD has been integrated in OpenCV.js, there are some remain tasks. Firstly, due to the limitation of emscripten at that time, the type 64 intrinsics for OpenCV.js does not utilize the SIMD api. Secondly, some kernels like `cvtColor` and `resize` do not achieve ideal results. Thirdly, the perf test only supports 3 imgproc kernels and lots of redundant code in perf test can be reused.

Now, with the updated emscripten, there are possibilities to solve the above problems. In this project, the major tasks are: 1. Implement the type 64 intrinsics with SIMD functions. 2. Try to use the new SIMD function to improve `resize` kernel. 3. Reuse the code in the perf test, design a clean perf test pattern and extend more perf tests for imgproc kernels. 4. Create a loader to detect the features of the browser and load corresponding OpenCV.js.

Work structure
Implement the type 64 intrinsics.
WebAssembly is adding the support of SIMD128 instructions (i.e., [WebAssembly SIMD](https://github.com/WebAssembly/simd/blob/master/proposals/simd/SIMD.md)). This feature has been landed in V8/Chromium behind a developer flag. Now WebAssembly SIMD has been utilized as a backend of OpenCV Universal intrinsics implementation by using LLVM WebAssembly builtins and WebAssembly intrinsics.

The WebAssembly intrinsics have implemented the 64bit intrinsics now, where no 64 intrinsics were implemented before Aug 2019. In the project, some fallback implementation used in the f64, i64, u64 universal intrinsics is replaced by the SIMD functions. The supported 64bit intrinsics contain `v_float43x2`, `v_in64x2`, `v_uint64`, `v_absdiff()`, `v_lut()`,

`v_shl()` and so on. To show the impact brought by 64bit intrinsics, the perf test for the function `Mat::dot()`, `split()`, `merge()` and `countNonZero()` for `CV_64F` is added.

Because the promise based module introduced in emscripten 1.39.16, a way to introduce the class `cv` asynchronously is added in the project asynchronously during the implementation of 64bit intrinsics.

Experiment on the kernel resize.

In the previous investigation, the SIMD optimization of the kernel `cvtColor` and `resize` had a big gap compared with Native SIMD optimization. For the `resize`, the biggest reason is that the `integer widening` instructions is simulated by the `shift` operation in the `v_dotprod`. In the emscripten 1.39.16, the [Integer to integer widening](https://github.com/WebAssembly/simd/blob/master/proposals/simd/SIMD.md#integer-to-integer-widening) is supported. In the process of this project, the `widen` instructions are tried and the performance data is collected. The performance data shows that the implementation with `widen` instructions does not bring improvement. In the implementation with `widen` instructions, the `wasm_v8x16_shuffle` and `widen` instructions have to be used together. It may be one of the reason why the performance has not improved.

Optimize the perf test and add more perf tests.
[Beachmark.js](https://benchmarkjs.com/) is a benchmarking library that supports high-resolution timers & returns statistically significant results. It is also the base of the original perf tests. However, the original perf tests have lots of replicated code. For example, all of the three original perf tests have a function called `setBenchmarkSuite()` and this function play a very similar role in these perf tests -- print the result of the perf test in a standard format. The `setBenchmarkSuite()` in different perf tests have limited differences. There is a large space to reuse it.

In this project, the redundant functions are removed in all perf tests and they are reconstructed as the common APIs in a single js file. It cleans the perf tests. During the optimization, a simple pattern for adding the perf tests is appeared by using these common APIs. In this pattern, developers just need to deal with designing kernel test cases and parsing input parameters. Based on this pattern, more perf tests for kernels including `Sobel`, `filter2D`, `scharr`, `gaussianBlur`, `blur`, `medianBlur`, `erode`, `dilate`, `remap`, `warpAffine`, `warpPerspective` and `pyrDown`. These perf tests are designed by referring the native perf tests for OpenCV. All performance data with SIMD and threads optimization of these kernels are collected and compared with the navtive SSE2 optimization and threads optimization.

Create a loader for OpenCV.js

OpenCV.js would have multiple builds for different browser features: ASM.js vs WASM, threads vs no threads, SIMD vs no SIMD. And the browser may or may not support each of these features. This phenomenon leads the requirement of the loader for OpenCV.js.

In this project, the loader is implemented as a js file in the path `<opencv_js_dir>/bin/loader.js`. The loader utilizes the [WebAssembly Feature Detection](https://github.com/GoogleChromeLabs/wasm-feature-detect) to detect the features of the browser and load corresponding OpenCV.js automatically. To use it, you need to use the UMD version of [WebAssembly Feature Detection](https://github.com/GoogleChromeLabs/wasm-feature-detect) and introduce the `loader.js` in your Web application:

```
<script src="https://unpkg.com/wasm-feature-detect/dist/umd/index.js"></script>
<script src="../../loader.js"></script>
```

The core function of the loader is the function `loadOpenCV()`. To use it, you need to construct the path configuration to point out the paths of each version of OpenCV.js. Then the path configuration and the main function are used as input parameters of the `loadOpenCV()`. The following is the example code:

```
// Set paths configuration
let pathsConfig = {
    wasm: "../../build_wasm/opencv.js",
    threads: "../../build_mt/opencv.js",
    simd: "../../build_simd/opencv.js",
    threadsSimd: "../../build_mtSIMD/opencv.js",
}

// Load OpenCV.js and use the pathsConfiguration and main function as the params.
loadOpenCV(pathsConfig, main);
```

Result

Perf test for 64bit intrinsics
OS: Ubuntu Linux 18.04.4

Emscripten: 1.39.16, LLVM upstream backend

Browser: Chrome, Version 85.0.4183.26 dev (64-bit)

Hardware: Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz with 12 logical cores

Function	Before 64bit implementation (ms)	After 64bit implementation (ms)	Speedup
countNonZero()	0.5386	0.5221	1.032x
Mat::dot()	0.8079	0.7960	1.015x
split()	2.1213	2.0483	1.036x
merge()	2.2383	2.2264	1.005x

The performance of kernels after 64-bit implementation is similar to the performance before implementation.

Performance of resize kernel with widen instructions

OS: Ubuntu Linux 18.04.4
Emscripten: 1.39.16, LLVM upstream backend
Browser: Chrome, Version 85.0.4183.26 dev (64-bit)
Hardware: Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz with 12 logical cores

Parameters	Mean time of Scalar	Mean time of SIMD with `shift`(ms)	Speedup	Mean time of SIMD with `widen`(ms)	Speedup
(CV_8UC4, 1280x720, 640x480)	5.113	2.888	1.77x	4.031	1.27x
(CV_8UC1, 1280x720, 640x480)	1.870	1.128	1.66x	1.249	1.50x

The perform of the implementation with `widen` instructions does not bring improvement. One of the reasons is that the `wasm_v8x16_shuffle` and `widen` instructions have to be used together. The original implementation is left finally.

Perf tests

Supported Kernels now (lines)	Supported Kernels before (lines)
cvtColor (421)	cvtColor (572)
resize (165)	resize (262)
threshold (158)	threshold (217)
Sobel (170)	-
filter2D (127)	-
Scharr (156)	-
gaussianBlur (126)	-
blur (130)	-
medianBlur (118)	-
erode (117)	-
dilate (117)	-
remap (182)	-
warpAffine (130)	-
warpPerspective (143)	-
pyrDown (116)	-

The collected performance data is stored in the Google Drive. [Performance data](https://docs.google.com/spreadsheets/d/1lVPdlLLSET1WT7yJZquSLcBgDcVtaExAbFk6Boxwjoo/edit?usp=sharing)

Future Work

* Some fallback implementations still exist in the intrinsics that utilizes the WebAssembly SIMD. With the update of emscripten and SIMD instructions, these fallback implementations would be replaced in the future.

* Besides the imgproc module, the perf tests for other modules like dnn cloud be added.

Perf and Loader Example

[Perf Test Example]()

[Loader Example]()

Commits List

[The PR](https://github.com/opencv/opencv/pull/18068)

[The list of my commits](https://github.com/opencv/opencv/pull/18068/commits/b201371bdfb10ab6004810ad26f3f40132b91ffe)