Implement light-weight Halide or Halide-like compiler and integrate it into OpenCV

- Author: Vadim Pisarevsky
- Link: The feature request
- Status: **Draft**
- Platforms: **All**
- Complexity: several man-months to 1-2 man-years.

Introduction and Rationale

Halide has been a revolutionary technology for accelerated array processing, co-developed by MIT, Stanford, Adobe and Google. It includes the notion of stencil operator, expressed in a form of Halide function, which is implicitly propagated to the whole processed array. There are quite elegant solutions for the border extrapolation (users just need to specify the extrapolation type and everything else is done automatically) and optimization of the processing pipelines, where intermediate results are computed and cached on demand. Essentially, an array processing pipeline (e.g. image processing pipeline) in Halide is expressed as superposition of functions. This superposition of functions is then translated to the actual CPU or GPU code. Users have fine control over the generated code, including tiling, vectorization, parallelization, loop unrolling, loop transposition etc. It's done by providing something called "schedule". This explicit separation of the math-like expression of the algorithm in terms of functions and the "close-to-metal" platform-specific schedule has been one of the main novel, revolutionary features of Halide.

Halide was proven to be useful for image processing acceleration; we've also got some promising results using Halide for OpenCV DNN module at the early optimization phase of this project. There are new perspective projects, like tvm that started with Halide engine and transformed it to very efficient deep learning inference engine.

In OpenCV Halide support can be useful for various reasons, such as:

- automatic generation of GPU/OpenCL code for regular kernels. Currently we write this code manually; implementing more kernels in OpenCL is very time consuming task.
- automatic generation of the platform-specific vectorized code, including AVX2 and AVX-512 instructions. This is another big problem in OpenCV - we have very few AVX2 and AVX-512 kernels, and adding more is straight-forward but time-consuming task
- dramatic reduction of the library footprint because of the automatic on-fly generation of the kernels. Currently in OpenCV the core and imgproc modules take most of the space, because of many populated versions of the same basic algorithms for all data types: at maximum OpenCV supports u8, s8, u16, s16, s32, f32, f64 data types. And in some functions a separate code is needed to handle 1-channel, 3-channel, 4-channel etc. data. If we can generate such code on-fly from the single Halide source, it would significantly reduce the binary size.
- fusion of simple kernels into efficient pipelines. Many basic functions in OpenCV not only take a lot of space, they are often sub-efficient, because the pure computing time is comparable with data transfer time. In other words, if one needs to construct a well-optimized pipeline, those basic functions are likely eliminated and replaced with manually fused loops. Halide would do most of this things automatically.

Proposed solution

Halide is open-source software under BSD license, just like OpenCV, so in theory we could just take it and put inside OpenCV. But Halide is:

1. quite heavy, Halide.DLL/.so takes ~50Mb
2. has serious limitations as a language; so it makes sense to slightly extend it (e.g. add if-operator or while-operator or explicit loops over pixels) in order to increase the amount of functionality that can be

implemented using Halide.
3. is not quite convenient at the interface level, e.g. how to pass actual parameters to the compiled Halide functions. The Python API is even less convenient and complete.
4. not quite efficient at the low level. Its loop fusion capabilities help it to achieve decent performance on the pipelines, but certain things, like color data processing or saturation arithmetics or OpenCL code generation are quite far from being ideal.
5. not quite efficient for DL inference as well. tvm folks say the same when they explain why they do not use Halide as-is; they take it as a starting point and started evolving the engine.

So the idea is to fork Halide, strip off unnecessary for us backends and other pieces, improve the rest, add lightweight CPU code generator (e.g. based on xbyak) and then integrate the result into OpenCV. Of course, it will be yet another fork, yet another fragmentation of open-source software world, but even because of item 1. we cannot use vanilla Halide. Of course, at some point pieces of our fork could be contributed back to the main Halide.

Impact on existing code, compatibility

It's the new functionality, so it should not affect the existing code. It may seriously affect the new OpenCV coding style though. Instead of using regular OpenCV functions the users may be advised to generate and compose Halide or Halide-like functions. But this is just for basic image processing and array processing stuff. Regular heavy algorithms, such as camera calibration, image stitching, optical flow, object detection etc. will remain the same externally.

Possible alternatives

Halide compiler embedded into OpenCV may be too complex thing to do and to support. Instead, we can consider something like a bit higher-level cross-platform Xbyak (think of dynamically generated universal intrinsics with) and reimplement some basic kernels using it. This way we can shrink the binary size and get retargetable dynamically-generated code, but it might be too low-level for OpenCV users.

References

TBD