Adding AVX512 Support

- Author: Vadim Pisarevsky
- Link: https://github.com/opencv/opencv/issues/13891
- Status: Open
- Platforms: All
- Complexity: Several man-weeks to a few man-months

Introduction and Rationale

AVX512 instructions have been introduced a few years ago, first as a part of instruction set of the special accelerator Xeon Phi, and then they have been added to some workstation-class and server-class CPUs (Skylake-X). The upcoming Icelake CPUs, which will be used in a wide range of devices, from servers downto ultra-portable laptops, should have AVX512 on board, so in theory in a few years from now (2019 Feb) AVX512 should become as common as SSE4.x and AVX2 nowadays.

AVX512 offers 32 512-bit registers (vs 16 256-bit registers in AVX2), some advanced instructions, including extensive mask support etc. In principle, AVX512 should provide noticeable boost in performance, comparing to SSE4 and AVX2, or the same performance with lower power consumption, because there are roughly 2x less instructions needed to perform the same task, so CPU can operate at lower clock speed. All in all, it makes sense to prepare OpenCV for AVX512 era.

OpenCV 4.0 has partial support for AVX512. First of all, on x64 OpenCV uses IPPICV, which includes AVX512-optimized functions. Also, our CMake scripts and the dynamic dispatching mechanism support AVX512 (there are two options, AVX512F and AVX512_SKX) and there are a few places, e.g. convolution kernel in DNN module, that use it. Nevertheless, it's very far from being complete. We need much more, and we need to get there with minimal efforts.

Proposed solution

In OpenCV we have well established and time-tested Universal Intrinsics, which we converted to Wide Universal Intrinsics (W.U.I) in OpenCV 4.0. They support SSE2 (or later, up to SSE 4.2), AVX2, NEON (32-bit and 64-bit ARMv7 or later), VSX (64-bit PowerPC arch). It would be natural to add yet another backend, intrin_avx512.hpp, so that all existing loops that use W.U.I can be compiled as AVX512-vectorized loops.

Here is the approximate plan:

- add intrin_avx512.hpp, https://github.com/opencv/opencv/tree/master/modules/core/include/opencv2/core/hal. intrin_avx2.hpp could be used as the donor. As soon as intrin_avx512.hpp will be more or less ready, https://github.com/opencv/opencv/blob/master/modules/core/include/opencv2/core/hal/intrin.hpp should be extended to include AVX512 parts. As in the case of AVX2-SSE2, AVX512 should augment AVX2-SSE2, not replace it. In other words, we should still let people to write AVX2 or SSE2 code even if OpenCV is compiled with AVX512 baseline instruction set.

- In addition to the normal v_uint8x64 etc. types that will be modified copies of v_uint8x32 etc., we need to introduce extra v_mask_uint8x64 etc. types and the corresponding size-less synonyms v_mask_uint8 etc. In all other files we need to introduce such types as well:

  ```
  // intrin_avx512.hpp
  typedef __mmask64 v_mask_uint8x64;
  ...
  // intrin_avx2.hpp
  ```

```
typedef v_uint8x32 v_mask_uint8x32;
...
// intrin_neon.hpp
typedef v_uint8x16 v_mask_uint8x16;

// intrin.hpp
#if CV_SIMD512
typedef v_mask_uint8x64 v_mask_uint8;
..
#elif CV_SIMD256
typedef v_mask_uint8x32 v_mask_uint8;
...
#else
typedef v_mask_uint8x16 v_mask_uint8;
...
#endif
```

- When in intrin_avx512.hpp we define comparison operators that return the mask types:

  ```
  inline v_mask_float32x16 operator == (const v_float32x16& a, const
  v_float32x16& b) { ... }
  ```

  For uniformity, we could have modified other backends as well, but I suggest to retain those backends as-is, since there `v_mask_sometype == v_sometype`.

- In intrin_avx512.hpp we also need to introduce extra operations for logical operations on masks, for casting masks from one type to another; we will also need v_all() and v_any() intrinsics etc. The other backends will likely need no modifications either.

- We can then define `v_select()` functions in intrin_avx512.hpp that will use `v_mask_sometype` as the first argument.

- Then all the loops where comparison ops and/or `v_select()` are used need to be modified to use the new types `v_mask_sometype`. The natural way to find and check all such places is to build OpenCV with CPU_BASELINE=AVX512_SKX and do it regularly on one of buildbot slaves.

- Most vectorized loops should compile fine as soon as we implement `+, -, *, |, ^, <<, >>, vx_load, v_store, ...`. However, in the case of AVX512 the non-vectorized loop tails can take a considerable amount of time, e.g. when processing 8-bit images the loop tail could be run up to 63 times! And it can degrade the performance significantly when the image width (or the processed ROI width) is not divisible by 64. In this case it's recommended to rewrite some of the loops to use Halide "repeated tail processing" technique (see also split.cpp and merge.cpp in the core module, where the technique is used):

  ```
  int i = 0;
  const bool inplace_op = ...; // figure out whether the operation is in-place or
  not

  #if CV_SIMD
  const int vecsize = ...;
  for( ; i < width; i += vecsize ) {
      if(i + vecsize > len) {
          if( inplace_op || i == 0 ) break;
  ```

```
        i = len - vecsize; // we shift i backwards; in this case some pixels
                           // are processed twice, but instead of many
iterations of the scalar loop
                           // we just do one more iteration of the vector loop
- profit!
    }
    ... // vectorized code
}
#endif
for( ; i < width; i++ ) {
   ... // scalar code; it's only used to process the tail
       // in the case of inplace op, or when the whole width
       // of the processed image is too small (less than 512 bits)
}
```

Impact on existing code, compatibility

As mentioned above, all the loops that use comparisons and `v_select()` (and maybe some others) need to be slightly modified. Most loops that do just computations without selective logic should compile as-is. All the possible incompatibilities should be caught by OpenCV buildbot.

Since it's about internal optimizations, nothing will change from the user perspective. It's assumed that the main/default OpenCV compile mode is to include (or not) AVX512 branch (at least for the most important functions) using dynamic dispatching mechanism. So the produced OpenCV binaries will stay fully compatible with older hardware and can utilize AVX512 instructions when OpenCV is running on the modern AVX512-capable hardware.

Possible alternatives

As mentioned earlier, AVX512 already exists in some form. The introduction of AVX512 backend of W.U.I can be postponed, and we can just accelerate critical loops by direct use of AVX512 intrinsics, but it will take noticeably more efforts at the implementation and, what's also important, at the support phase. So it's rather "when" than "if" question.

References

- Intel guide to AVX512 intrinsics:
  https://software.intel.com/sites/landingpage/IntrinsicsGuide/#techs=AVX_512