Introduction

There are many processors architectures: x86 / x86-64 family, ARMv7, aarch64, etc. Each architecture may support different additional instruction sets (SSE/AVX for x86, NEON for ARMv7).

OpenCV goal is to provide effective processors support, including separate optimized code paths for newest instruction sets.

Some OpenCV functions contains multiple code paths specialized for different processors features / instruction sets. Selection of executed code path is based on auto-detection of available processor features.

Note: Build options described here don't control behavior of CPU-based optimizations from Intel® Integrated Performance Primitives (Intel® IPP, https://software.intel.com/en-us/intel-ipp).

Customizing CMake options

These options are available since OpenCV 3.3 (released in Aug 2017).

Build options allow to specify **minimal** and **dispatched** optimization features sets:

- **Minimal** is required set of processor features. Executable will not run if some of these options are not available on target processor.
- **Dispatched** optimizations are additional code paths compiled into executable. They will be executed on supported processors only.

By default, OpenCV on x86_64 uses SSE3 as basic instruction set and enables dispatched optimizations for SSE4.2, AVX, AVX2 instruction sets. This configuration provides the best effort on wide range of users platforms.

OpenCV uses these CMake variables to control supported optimization features:

- `CPU_BASELINE` - **minimal** set of required optimizations (if they are supported by C++ compiler)

  ```
  CPU_BASELINE=SSE2
  CPU_BASELINE=AVX
  ```

- `CPU_DISPATCH` - **dispatched** set of additional optimizations (if they are supported by C++ compiler)

  ```
  CPU_DISPATCH=SSE4_2,AVX
  CPU_DISPATCH=AVX
  CPU_DISPATCH=AVX,AVX2
  ```

Note: Flags `ENABLE_AVX` / `ENABLE_AVX2` / `ENABLE_POPCNT` /etc should not be used anymore. Use options above instead.

Advanced CMake options:

- `CPU_BASELINE_REQUIRE` - list of required baseline (minimal set) optimizations. Build fails if compiler doesn't support some of these optimizations.
- `CPU_DISPATCH_REQUIRE` - list of additional (dispatched set) optimizations. Build fails if compiler doesn't support some of these optimizations.
- `CPU_BASELINE_DISABLE` - list of completely disabled optimizations
- `CV_DISABLE_OPTIMIZATION` - disable all explicit optimized code (useful for debugging purposes)

CMake status of used optimizations:

OpenCV configuration status contains lines like these:

```
--   CPU/HW features:
--     Baseline:                    SSE SSE2 SSE3 SSSE3
--       requested:                 SSSE3
--     Dispatched code generation:  SSE4_1 AVX AVX2
--       requested:                 SSE4_1 AVX AVX2
```

Source files layout

In general, C++ compilers don't support code generation of multiple enabled instruction sets for single source file.

Current layout of source files is:

- main source file: "cvfunction.cpp" - contains code dispatcher (see below) and general implementation
- shared header file: "cvfunction.hpp" - contains shared declarations used by generic and optimized code
- several files with optimized code (suffix `.<lowercase optimization name>.cpp`): "cvfunction.avx.cpp".

There is no requirement to implement all possible optimizations. Add function optimizations with better effort only.

Code dispatcher

Unwrapped version:

```
CV_CPU_CALL_AVX2(my_cv_function_avx2(...args...));
CV_CPU_CALL_FP16(my_cv_function_fp16(...args...));
CV_CPU_CALL_AVX(my_cv_function_avx(...args...));
CV_CPU_CALL_SSE4_2(my_cv_function_sse4_2(...args...));
CV_CPU_CALL_POPCNT(my_cv_function_popcnt(...args...));
CV_CPU_CALL_NEON(my_cv_function_neon(...args...));

... regular C++ implementation ...
```

These macro declarations are depend on CMake options:

- empty statement (in case of non-used or disabled optimization)
- conditional code execution: `if(checkHardwareSupport(CV_AVX)) ...`
- unconditional code execution in case of baseline optimization

AVX optimization note

Mixing of AVX and SSE code may provide significant performance reduction due registers sharing. Work around for this is to use `_mm256_zeroupper()` intrinsic:

```
#if CV_AVX && !defined CV_CPU_BASELINE_COMPILE_AVX
    _mm256_zeroupper();
#endif
```

Caution about C++ templates

All non-inline templates used in optimized code must be wrapped into separate (or anonymous) namespaces to prevent conflicts (linker will use one template's implementation only).

Unfortunately, there is no any warning from linker about this.

Runtime environment variables

- `OPENCV_CPU_DISABLE` - allows to mask some processor features, so dispatched code doesn't run

  ```
  OPENCV_CPU_DISABLE=AVX2,AVX,FP16
  ```

Optimization developer guide

This "How to" example is based on optimization of Hamming norm algorithm (`core` module, file stat.cpp).

1. Ensure that you have performance tests for selected functionality. Please don't waste your time and time of reviewer doing this without good performance tests.
2. Compile OpenCV performance test with different CPU baseline features with disabled dispatching (depends on your platform). I select on `x86-64` platform: `SSE3` (minimal), `SSE4.1` , `SSE4.2` , `AVX` , `AVX2` (max level on my platform), `DETECT` (with `-march=native` compiler option). It is better to build these versions of OpenCV configuration in different folders. When run performance tests and build report like this: [[images/cpu_dispatch_0_baseline.png|Baseline performance]]
3. On this report we can see on the second part (with `--progress` option):
   - We would gain performance improvement on dispatched
     - SSE4.2 (`popcount` instruction, **`1.7-1.9`** speedup)
     - AVX for `norm` function only (**`1.3`** speedup over SSE4.2)
     - AVX2 (**`~3`** total speedup, **`~1.5`** speedup after SSE4.2/AVX)
   - Dispatching for SSE4.1 mode is useless.
   - Dispatching for `NORM_HAMMING2` will not increase speed, so we avoid it.
4. Let's extract implementations of interested functions into separate `.simd.hpp` file "as is". This header file will be processed multiple times - so we will generate binary code with different optimization options using single source file. Refer to PR commit "*move implementations into .hpp file w/o changes*".
5. After that we should "*create dispatch.cpp file*". It is simple helper file without algorithm logic, but it contains entry-points for optimized functions and dispatch rules.
6. We need to "*remove useless checks*" from `.hpp` file, because these platform dependent checks is done in compile-time (controlled via defines).
7. On the next step we should "*register dispatched code, fix build*".
   - We should register our dispatched code from `.hpp` file via CMake handler. Also we pass list of enabled optimizations: `SSE4_2 AVX AVX2` .
   - We reusing `popCountTable` multiple times from different compilations units, so we need to make it "external".
8. Mixing of SSE/AVX code during runtime usually provides significant performance impact. To workaround this problem we should use `vzeroupper` instruction. See commit: "*add required CV_AVX_GUARD*"
9. Build performance tests:
   - separate build directory
   - check for enabled dispatching levels: "-DCPU_DISPATCH=SSE4_2;AVX;AVX2"
10. Run performance tests:

   - without restrictions and save results to "avx2.xml"
   - "avx.xml": mask AVX2 optimizations via environment variable: `OPENCV_CPU_DISABLE=AVX2`
   - "sse42.xml": mask optimizations via environment variable: `OPENCV_CPU_DISABLE=AVX2,AVX`
   - "sse3.xml": mask optimizations via environment variable: `OPENCV_CPU_DISABLE=AVX,AVX2,SSE4.2`

11. Result is here: [[images/cpu_dispatch_2_dispatch_after.png|Dispatched Hamming norm]]

12. We can see that there is no improvements from dispatched AVX optimization, so we can remove it from CMake file: "*optimize size of binaries, drop AVX dispatching*"

    - AVX-related issue is a compiler problem with processing of unrolled loops. Removal of these loops before generation of the first report resolves this strange behavior (slowdown of SSE4.2 code).

13. Final result is here: [[images/cpu_dispatch_3_dispatch_after_drop_AVX.png|Final dispatched Hamming norm]]

For reference, similar report for code without patch: [[images/cpu_dispatch_1_dispatch_before.png|Dispatching of Hamming norm before patch]]

Additional possible changes:

- support `NORM_HAMMING2` too, but this will require additional optimizations (probably for AVX2 code only).
- we can replace SSE4_2 to POPCNT dispatch level in CMake file.