Running OpenCV on RISC-V CPUs

RISC-V is the relatively new platform with growing popularity, especially in the IoT market. In 2020 OpenCV has got support for this platform.

Efficient execution of various image processing, computer vision and machine learning algorithms requires vector/SIMD instructions, and many modern CPU architectures provide such instructions. In the case of RISC-V OpenCV relies on so-called V Extension (or RVV in short). At the moment of writing, specification of this extension is still not finalized. However, some tools (compilers, software emulators) are already available and the real hardware based on some draft versions of the standard is being prepared for launch.

Whereas different CPU architectures have different vector instructions, many time-critical kernels in OpenCV are optimized using our own cross-platform Wide Universal Intrinsics — light-weight inline functions with uniform cross-platform API and platform-specific implementations (that use native intrinsics, e.g. SSE/AVX intrinsics on x64 platform, NEON intrinsics on ARM etc.). You may find the corresponding implementations inside the core module: https://github.com/opencv/opencv/tree/master/modules/core/include/opencv2/core/hal. In the case of RISC-V we provide 2 parallel implementations, `intrin_rvv.hpp` (based on the latest draft of the standard, provided by the Chinese Academy of Sciences) and `intrin_rvv071.hpp` (contributed by T-head company (平头哥半导体有限公司): https://www.t-head.cn).

Building OpenCV for T-head's Xuantie-910 CPU:

1. Clone `opencv` and `opencv_extra` repositories from https://github.com/opencv, the master branches. Let's assume, you put it to `$HOME/projects/opencv` and `$HOME/projects/opencv_extra`, respectively.
2. Download the toolchain and qemu emulator (needed for testing, unless you already have the real hardware) from https://github.com/damonyu1989/Tools
3. Unpack the tools, put them in some directory, e.g. /opt/rvv071, and add /opt/rvv071/bin to the PATH (e.g. in `~/.bashrc`). At once, set `OPENCV_TEST_DATA_PATH` environment variable:

```
PATH=/opt/rvv071/bin:$PATH
export OPENCV_TEST_DATA_PATH=$HOME/projects/opencv_extra/testdata
```

4. Configure OpenCV (tested on Ubuntu 20.04 x64):

```
mkdir $HOME/projects/build/opencv_rvv071
cd $HOME/projects/build/opencv_rvv071
cmake ../../opencv -G "Unix Makefiles" -
DCMAKE_TOOLCHAIN_FILE="../../opencv/platforms/linux/riscv64-0.7.1-
gcc.toolchain.cmake" -DWITH_OPENCL=OFF -DBUILD_opencv_calib3d=ON -
DBUILD_ZLIB=ON -DBUILD_PNG=ON -DBUILD_SHARED_LIBS=OFF -
DCMAKE_BUILD_WITH_INSTALL_RPATH=1 -DCMAKE_INSTALL_PREFIX=./install
make -j8
make install # optionally
```

5. You can now run the tests on the real hardware or QEMU emulator. In the latter case, you can do it with

```
cd bin
qemu-riscv64 ./opencv_test_core  # run core tests, you can also run imgproc,
dnn etc. tests
```