OpenCV Introduction and New Features

OpenCV China Team

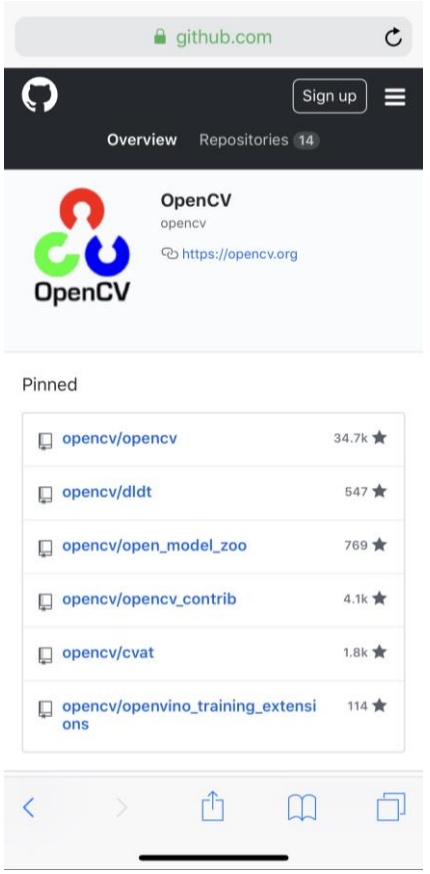
Outline

- General overview
- What's new in OpenCV 4.x
- OpenCV DNN module: overview & new features
- How to make OpenCV run fast

OpenCV is	The most popular “traditional” computer vision library with growing DL capabilities: http://opencv.org
License	**BSD** (free for non-commercial and commercial use)
Supported Languages	**C/C++**, Java, **Python**, Javascript
Size	**>1M** lines of code
SourceForge statistics	**20M** downloads (not include github traffic)
Github statistics	>10K clones per week, >**9000** patches merged since 2012 *(>5 patches per day)*
Accelerated with	**SIMD:** SSE, AVX2, AVX512, NEON**, many core CPUs:** parallel for, **GPU:** OpenCL, CUDA, Vulkan**, S/W:** IPP, MKL, Intel DLDT
The actual versions	**3.4.9, 4.2.0** (Dec. 2019)

More than just OpenCV

opencv	The main OpenCV repository, essential, stable modules
opencv_contrib	Experimental or obsolete OpenCV functionality
cvat (Computer Vision Annotation Tool)	Tool for annotation of datasets; reworked version of VATIC
dldt (Deep Learning Deployment Toolkit)	Very fast Deep Learning Inference Engine and Model Optimizer/Converter tool; for Intel/AMD platforms only.
open_model_zoo	High-quality CV deep learning models by Intel
opencv_training_extensions	Scripts for TensorFlow, PyTorch etc. to retrain some of the models from **open_model_zoo**, quantize networks etc.



http://github.com/opencv

OpenCV History

2007. **OpenCV Symposium in Beijing,** organized by **Shiqi Yu, Vadim Pisarevsky and Shinn-Horng LEE.**

2006. OpenCV China web site

2006. OpenCV 1.0 gold is finally out. **5** modules (core, cv, cvaux, ml, highgui), **~200K** lines of code, **500+** functions & classes (ml), HTML docs, no IPL dependency, uses **IPP** for acceleration, **Win & Linux, x86 & x64** support, includes Python interface

2000. OpenCV 1.0 alpha announced at CVPR. **Win32** only; C API; includes image processing, contours, LK optical flow, … uses **IPL** as complimentary library

2000. **Vadim Pisarevsky** joined OpenCV Development Team as the team leader.

1998. OpenCV project started at Intel under name **CVL** by **Gary Bradski**

itseez

Willow Garage

magic leap

intel

NVIDIA

AMD

Google

GitHub

OpenCV: Itseez!

2015. **OpenCV 3.0**. **30+30** modules (with **opencv_contrib**), **>1M** lines of code, **T-API**, **NEON & AVX** accelerations, free **IPP**, **GSoC 2013 & 2014** results

2012. **OpenCV 2.4.0-2.4.3**. **22** modules, **~600K** lines of code, automatically generated Python & Java interfaces, **CUDA & OpenCL** acceleration, lot's of parallel code, **GSoC 2011&2012** results, **Win, Linux, Android, iOS, x86, x64, ARM** support. Migrated to github

2009. OpenCV 2.0. Rewritten in C++, uses CMake, lots of native **SSE2** accelerations. LBP face detection, HOG pedestrian detector, Farneback dense optical flow, FAST corner detector, MSER region detector.

OpenCV: Back to Intel!

2018. **OpenCV 4.0**. *Major cleanup. Emphasis on DL. Halide integration. Intel* OpenVINO *integration.*

2016. **OpenCV 3.2**. Deep Learning module. OpenVX support. **GSoC 2015 & 2016** results: state-of-art traditional optical flow, stereo, object detection, text detection, feature detection, tracking, calibration, comp. photography algorithms.

Plan to


OSVF


OpenCV China


Teams in Russia

OpenCV Distributed Team (?)

2025

- …
- OSVF, OpenCV China Foundation (to be founded) support OpenCV
- OpenCV Courses (online+offline) in multiple languages, especially in Chinese
- Better support for 3D sensors (lidar, ToF camera)
- Better support on ARM and specialized DL acclerators (NPU)
- **2020 OpenCV 20-year anniversary and OpenCV 5 Release**
- 2019-… OpenCV 4.x Release

Developers and Contributors

USA, Bay Area

intel

Google

Google Summer of Code

Open Source Vision Foundation (OSVF); OpenCV.org

China, Shenzhen

OpenCV.org.cn (since 2019 Dec)

AIRS SHENZHEN INSTITUTE of ARTIFICIAL INTELLIGENCE AND ROBOTICS for SOCIETY 深圳市人工智能与机器人研究院

OPEN AI LAB

Russia, Ni.No

intel

Core OpenCV team @ Intel Russia

OpenVINO

XPERIENCE.AI

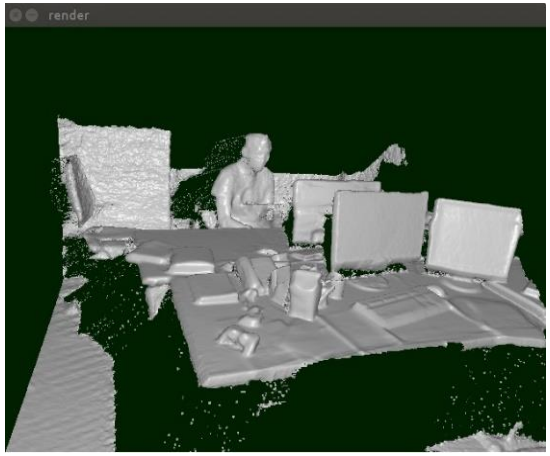
Community

2-3 patches per day via github.com

Major New Features of OpenCV 4.x

- C++ 11 library!
- Emphasis on Deep Learning (see further)
  - Significantly extended and accelerated OpenCV DNN module
  - Started replacing some traditional algorithms in OpenCV with deep nets (e.g. face, object, text detection)
- Introduced graph API (G-API) for efficient image processing pipelines
- Smaller and faster
  - AVX2 & AVX512 acceleration; NEON acceleration for 32-bit and 64-bit ARM CPUs; ~10-30% acceleration using AVX2!
  - Lower footprint. OpenCV 4.0 is ~20% smaller than OpenCV 3.x.

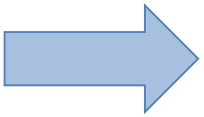
- FP16 support (especially useful for efficient Deep Learning inference): **cv::Mat fp16_tensor({32,32,16}, CV_FP16);**
- Video I/O: Hardware-accelerated video decoding/encoding on Windows (WMF) and Linux (GStreamer), new Android backend …
- QR code detector and decoder
- Results from GSoC 2017 and GSoC 2019



Graph API (G-API) Overview

- A new separate module opencv_gapi (not a complete library rewrite): https://github.com/opencv/opencv/wiki/Graph-API
- Provides alternative “lazy” image processing functions, e.g. cv::Sobel => cv::gapi::Sobel
  - Instead of immediate evaluation gapi:: functions construct expression subgraphs (GMat) and then you get a complete graph (GComputation)
- The produced graph is compiled (once) and then can be processed more efficiently than a sequence of direct function calls
- CPU and GPU backends are ready; more backends are in progress

```cpp
#include "opencv2/core.hpp"
#include "opencv2/imgproc.hpp"

#include "opencv2/highgui.hpp"

int main(int argc, char *agrv[])
{
    using namespace cv;
    if (argc != 3) return 1;

    Mat in_mat = imread(argv[1]);
    Mat gx, gy;

    Sobel(in_mat, gx, CV_32F, 1, 0);
    Sobel(in_mat, gy, CV_32F, 0, 1);

    Mat mag, out_mat;
    sqrt(gx.mul(gx) + gy.mul(gy), mag);
    mag.convertTo(out_mat, CV_8U);

    imwrite(argv[2], out_mat);
    return 0;
}
```



```cpp
#include "opencv2/gapi.hpp"
#include "opencv2/gapi/core.hpp"
#include "opencv2/gapi/imgproc.hpp"
#include "opencv2/highgui.hpp"

int main(int argc, char *agrv[])
{
    using namespace cv;
    if (argc != 3) return 1;

    GMat in;
    GMat gx = gapi::Sobel(in, CV_32F, 1, 0);
    GMat gy = gapi::Sobel(in, CV_32F, 0, 1);
    GMat mag = gapi::sqrt(gapi::mul(gx, gx),
                          + gapi::mul(gy, gy));
    GMat out = gapi::convertTo(mag, CV_8U);
    GComputation sobel(in, out);

    Mat in_mat = imread(argv[1]), out_mat;
    sobel.apply(in_mat, out_mat);
    imwrite(argv[2], out_mat);
    return 0;
}
```

G-API: Print imaging benchmark

We can see the “Graph effect” :

- Memory consumption – process big images by tiles w/o storing intermediate results explicitly
- Cache efficiency => better efficiency
- Code compactness – better performance with no need to write custom “fused” loops
- [to be added soon] automatic offloading to GPU

Memory consumption*

Input	OpenCV MiB	G-API/Fluid MiB	Factor Times
512 x 512	17.33	0.59	28.9x
640 x 480	20.29	0.62	32.8x
1280 x 720	60.73	0.72	83.9x
1920 x 1080	136.53	0.83	164.7x
3840 x 2160	545.88	1.22	447.4x

Performance* (based on cache efficiency)

Input	OpenCV ms	G-API/Fluid ms	Factor Times
320 x 240	1.16	0.53	2.17x
640 x 480	5.66	1.89	2.99x
1280 x 720	17.24	5.26	3.28x
1920 x 1080	39.04	12.29	3.18x
3840 x 2160	219.57	51.22	4.29x

* – all measurements are taken on Intel® Core™-i5 6600 CPU, single thread

DNN module overview

- Compact self-contained implementation in C++; inference only!
- 5 importers (Caffe 1, TensorFlow, Torch, Darknet, ONNX)
- 40+ layers, 100+ unit tests, 20+ samples
- Supports many popular topologies: image classification, object & text detection, semantic segmentation, instance segmentation, pose estimation, face recognition, style transfer, tracking, etc.

- Easy-to-use C++, Python, Java and Javascript interface

```
Net net = readNet(model_name, model_config);
Mat blob = blobFromImage(img, ...);
net.setInput(blob);
Mat out = net.forward();
```

- Sophisticated layer fusion mechanism & memory manager to improve efficiency and decrease memory footprint
- Many different execution backends with graceful fallback to the default C++ implementation:

Backend	CPU	iGPU fp32	iGPU fp16	dGPU	Intel VPU (NCS/NCS 2)	FPGA
DNN_BACKEND_OPENCV	+	+	+	+	–	–
DNN_BACKEND_INFERENCE_ENGINE	+	+	+	–	+	+
DNN_BACKEND_HALIDE (Deprecated)	+	+	–	+	–	–
DNN_BACKEND_VKCOM (Vulkan)	–	+	–	?	–	–
DNN_BACKEND_TENGINE (soon)						

Supported Topologies

- Classification

  Caffe: AlexNet, GoogLeNet, VGG, ResNet, SqueezeNet, DenseNet, ShuffleNet

  TensorFlow: Inception, MobileNet

  Darknet (https://pjreddie.com/darknet/imagenet/), ONNX (https://github.com/onnx/models)

- Object detection

  Caffe: VGG-SSD, Mobilenet-SSD, Faster-RCNN, R-FCN

  TensorFlow: SSD, Faster-RCNN, Mask-RCNN (TF OD API), EAST text detection

  YOLOv2, TinyYOLO, YOLOv3 (all Darknet), TinyYOLOv2 (ONNX)

- Semantic segmentation

  FCN (Caffe), ENet (Torch), ResNet101_DUC_HDC (ONNX)

- Other

  OpenPose body and hands pose estimation (Caffe), Colorization (Caffe), Fast-Neural-Style (Torch), OpenFace face recognition (Torch)

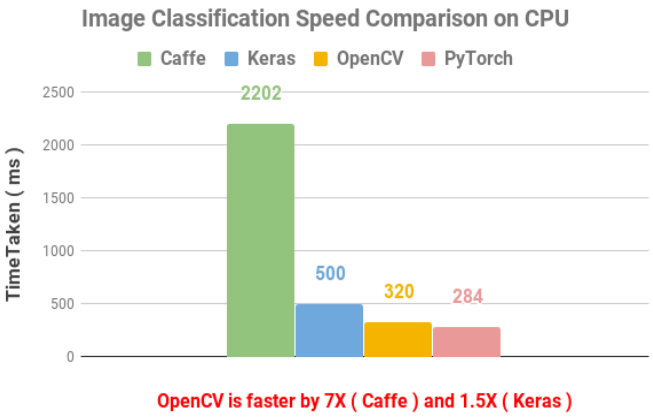
Refer to https://github.com/opencv/opencv/wiki/Deep-Learning-in-OpenCV for details

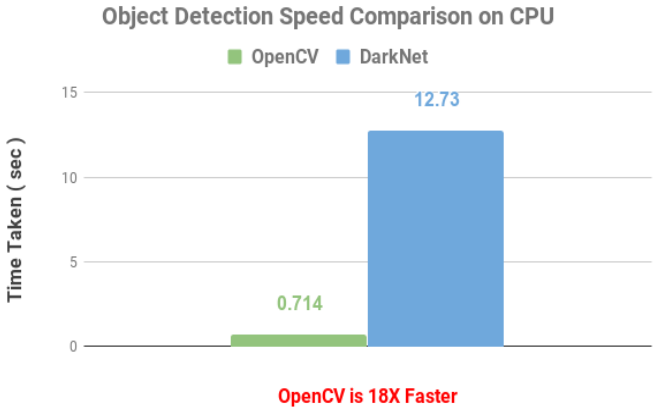
SSD-like network versus Haar Cascades

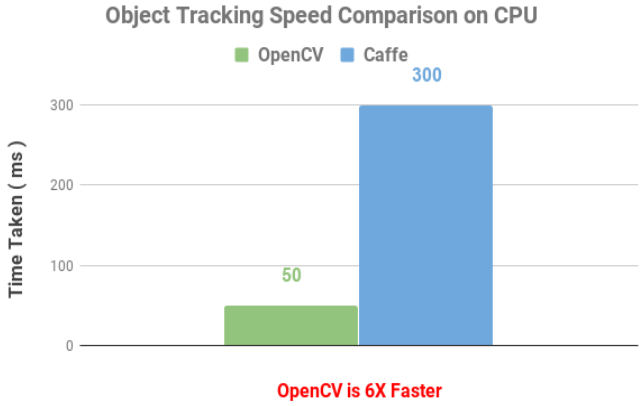
	Haar Cascade	DL
Size on disk	528KB	10MB (fp32), 5MB (fp16)
Efficiency @ 300x300**	30 ms	9.34 ms
Performance AP @ IoU = 0.5*	0.609 (FDDB) 0.149 (WIDER FACE, val.)	0.797 (FDDB) 0.173 (WIDER FACE, val.)

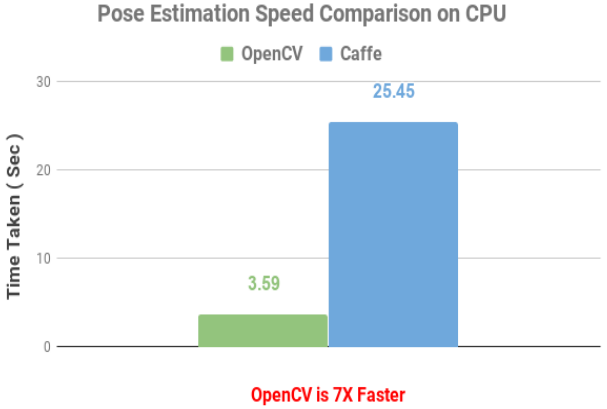
*PASCAL VOC metric using COCO evaluation tool, http://cocodataset.org/#detections-eval

**Intel® Core™ i5-4460 CPU @ 3.20GHz x 4







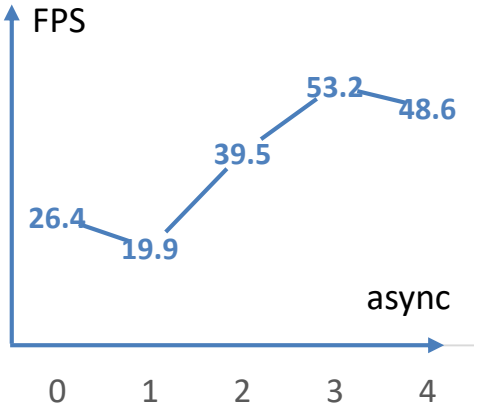


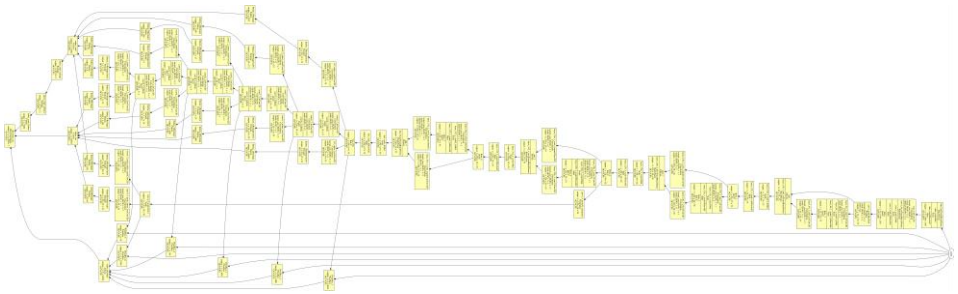
https://www.learnopencv.com/cpu-performance-comparison-of-opencv-and-other-deep-learning-frameworks/

New Features in OpenCV DNN (4.x)

- Vulkan-based backend (for Android)
- CUDA-based backend (GSoC 2019): https://github.com/opencv/opencv/pull/14827
- Intel NCS and NCS2 support via Intel Inference Engine
- ONNX importer added in 4.0, extended in 4.1.x
- Mask-RCNN topology support + mask_rcnn.py sample
- 3D CNNs support. New Action Recognition sample: action_recognition.py
- New high-level API for detection, semantic segmentation
- Asynchronous inference
- Deep learning networks visualization: cv::dnn::dumpToFile(dot_file);
- Improvements of ONNX and TensorFlow importers
- 18% speedup of YOLOv3 on NCS2

async face detection performance on Intel NCS2





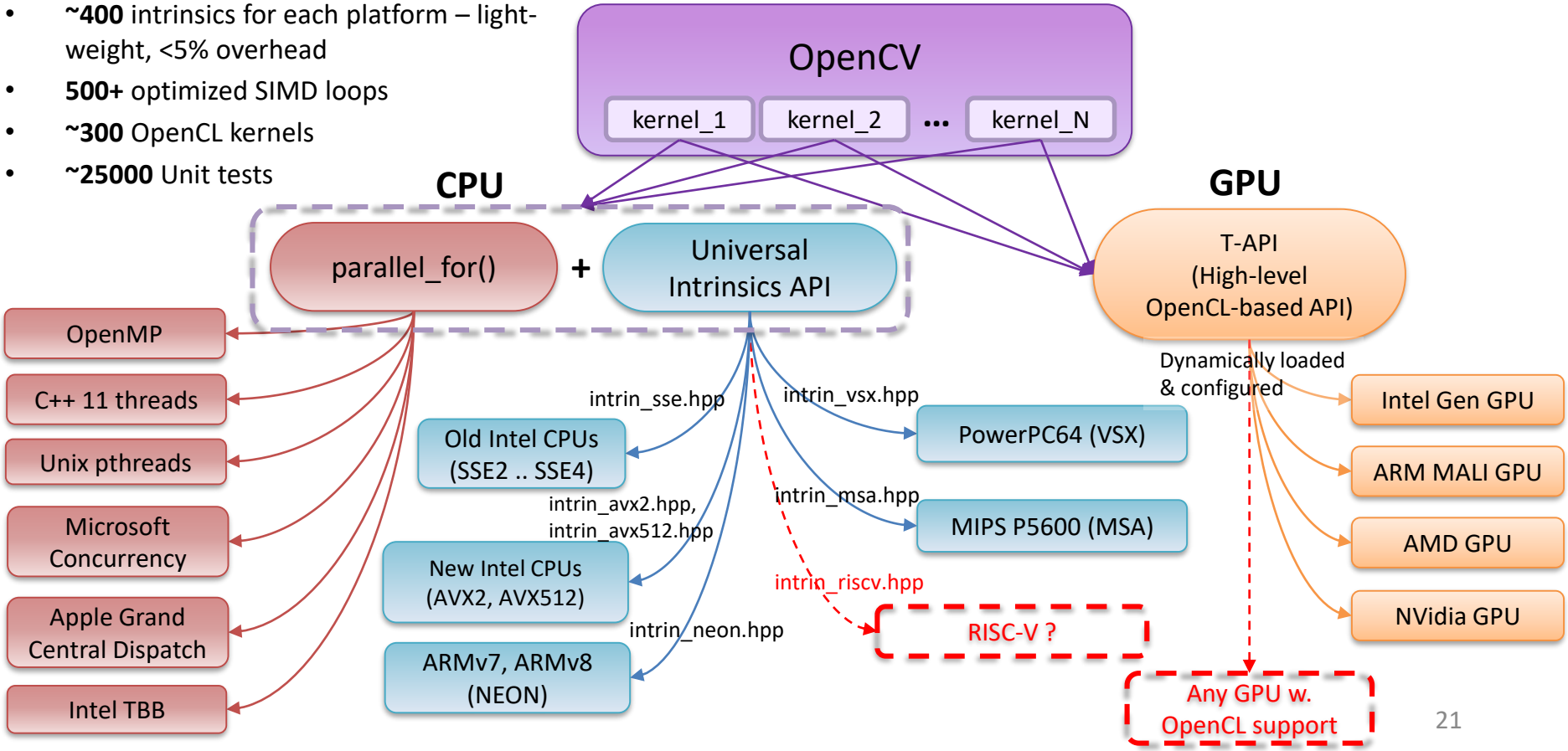
dump of opencv_face_detect model (shows fused layers)

Acceleration on Different Platforms

Tools/libs	Applicable for	
cv::parallel_for_	many-core CPUs	
wide universal intrinsics	CPUs with SIMD (vector) instructions. *Dynamic dispatching is used since OpenCV 4.0*	Intel & AMD x86/x64: SSE2-4, AVX2, AVX512
		ARM v7 and v8 (aarch64): NEON
		PPC64: VSX
		MIPS: MSA (PR submitted)
OpenCL (OpenCV T-API)	Intel iGPU, AMD GPU, Nvidia GPU	
CUDA	NVidia GPU (deprecated, except for DNN)	
Vulkan	DNN Inference on GPU (mostly for Android)	
IPP, MKL, OpenBLAS	CPU (traditional vision; image processing & linear algebra)	
Intel DLDT	DNN Inference on Intel CPUs, GPUs, VPUs	
Tengine	In progress: DNN Inference on ARM	

write once, run ***fast*** everywhere

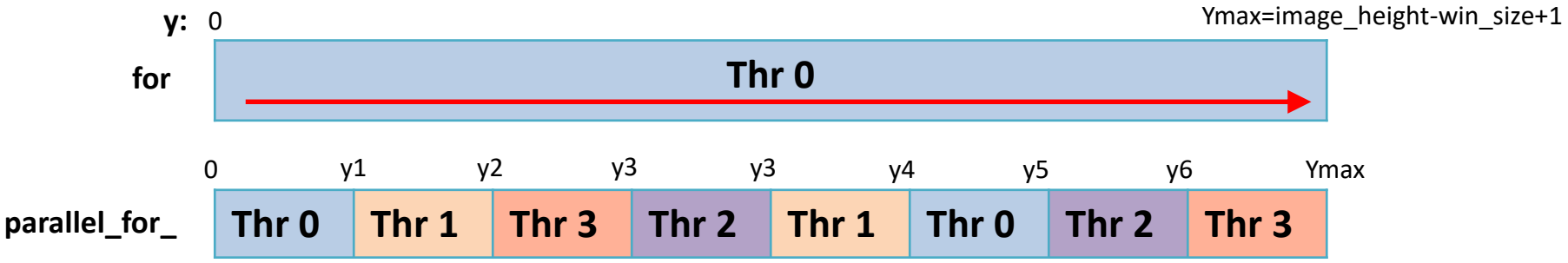
- **~400** intrinsics for each platform – light-weight, <5% overhead
- **500+** optimized SIMD loops
- **~300** OpenCL kernels
- **~25000** Unit tests



21

Accelerating OpenCV on CPUs: cv::parallel_for_

- cv::parallel_for_ – cross-platform implementation of parallel loop concept (uses Win32 threads, std::threads, GDC, pthreads, OpenMP etc. underneath).
- cv::Mutex – cross-platform implementation of thread synchronization object



Accelerating OpenCV on CPUs: wide universal intrinsics

```
// 3x3 image blur: scalar version
Mat src = imread("lena.jpg"), dst(src.size(), src.type());
int cn = 3, scale = cvRound((1./9)*(1 << 16)), step = (int)src.step;
for( int y = 1; y < image_height - 1; y++ ) {
   uchar *sptr = src.ptr<uchar>(y), *dptr = dst.ptr<uchar>(y);
   for( int x = cn; x < (image_width - 1)*cn; x++, sptr++ ){
      int s = sptr[-step-cn] + sptr[-step] + sptr[-step+cn] +
              sptr[-cn] + sptr[0] + sptr[cn] +
              sptr[step-cn] + sptr[step] + sptr[step+cn];
      dptr[x] = (uchar)(s*scale >> 16);
   }
}
```

```
// 3x3 image blur: vectorized version
#include "opencv2/core/hal/intrin.hpp"

int cn = 3, scale = cvRound((1./9)*(1 << 16)), step = (int)src.step;
const int VECSZ = v_uint16::nlanes;
v_uint16 v_scale = vx_setall_u16(scale);
for(int y = 1; y < img.rows-1; y++) {
   uchar *sptr = src.ptr<uchar>(y), *dptr = result.ptr<uchar>(y);
   int x = cn;
#if CV_SIMD // the vector loops expands to SSE/AVX loop on Intel/AMD, NEON loop on ARM etc.
   for( ; x <= (img.cols-1)*cn - VECSZ; x += VECSZ, sptr += VECSZ) {
      v_uint16 s = vx_load_expand(sptr-cn) + vx_load_expand(sptr) + vx_load_expand(sptr+cn) +
                   vx_load_expand(sptr-step-cn) + vx_load_expand(sptr-step) + vx_load_expand(sptr-step+cn) +
                   vx_load_expand(sptr+step-cn) + vx_load_expand(sptr+step) + vx_load_expand(sptr+step+cn);
      v_pack_store(dptr + x, v_mul_hi(s, v_blur_scale));
   }
#endif
   for( /*int x = cn*/; x < (image_width - 1)*cn; x++, sptr++ ){
      int s = sptr[-step-cn] + sptr[-step] + sptr[-step+cn] +
              sptr[-cn] + sptr[0] + sptr[cn] +
              sptr[step-cn] + sptr[step] + sptr[step+cn];
      dptr[x] = (uchar)(s*scale >> 16);
}}
```

The Future

- **OpenCV on Edge** – much better **ARM** support:
  - more optimizations of traditional vision algorithms
  - DNN inference optimization (probably, using Tengine by OpenAI)
    - Support for specialized H/W DNN accelerators
  - extensive testing (Continuous Integration) of OpenCV on **ARM**
- Improved documentation, tutorials, courses (online & offline)
- Slow but steady refinement of traditional CV functionality (camera calibration, 3D vision, image processing …)
- **OpenCV 5**

Thank You !