Using Intel® IPP with OpenCV

Introduction

Intel® Integrated Performance Primitives (Intel® IPP) is an extensive library of multicore-ready, highly optimized software functions for digital media and data-processing applications. Intel® IPP offers thousands of optimized functions covering frequently-used fundamental algorithms. Intel IPP functions are designed to deliver performance beyond what optimized compilers alone can deliver. More information about IPP can be retrieved at http://www.intel.com/software/products/ipp/index.htm.

OpenCV is an acronym for Open Source Computer Vision Library. The library is a well-known software library in computer vision for both academic and commercial use. It is free, open source software and provides developers an easy way to input, display and store video and images, and also provides over 500 routines for computer vision processing, image processing, face detection and tracking, machine learning, etc. More information about OpenCV can be found at http://opencv.willowgarage.com/wiki/FullOpenCVWiki

In early OpenCV versions, OpenCV was automatically accelerated by taking advantage of Intel® Integrated Performance Primitives (Intel® IPP). However, the latest OpenCV version 2.1.0 doesn't incorporate Intel® IPP by default, thus the performance benefit of Intel® IPP functions which are optimized via the Streaming SIMD Extensions (SSE, SSE, SSE2, SSE3, SSSE3, SSE4, SSE4.1, SSE4.2, and Intel® AVX in latest) instructions, cannot be obtained automatically.

With the technical notes, we will show how to use Intel® IPP with latest OpenCV as well as how to integrate Intel® IPP into OpenCV. There are two test cases provided to demonstrate the details.

1. Use Intel® IPP library and OpenCV library independently
   Test case: transpose a image via cvRemap and ippiRemap
2. Integrate Intel® IPP library into OpenCV library and use OpenCV with built-in Intel IPP
   Test case: Measures image similarity via cvMatchTemplate() with default OpenCV library and OpenCV library with built-in-IPP
3. Compare OpenCV, OpenCV with built-in Intel® IPP, and Intel® IPP
   Test case: cvMatchTemplate() and ippiCrossCorrValid_Norm()

1. Use Intel® IPP library and OpenCV library independently

OpenCV is a collection of C functions and a few C++ classes that implement many popular Image Processing and Computer Vision algorithms. Intel® IPP is a collection of highly optimized functions for digital media and data-processing applications. There is some duplication of functionality between the two libraries. For specific information, please see some comparison in Intel® Ipp - Open Source Computer Vision Library (OpenCV) FAQ .

Engineers who develop image/video processing may often use the two libraries in the same application, for example to transpose a image from horizontal orientation to vertical orientation. In our test case we did this and implemented processing using OpenCV and Intel IPP.

Step 1. Install the OpenCV library,
Go http://www.sourceforge.net/projects/opencvlibrary to download the latest OpenCV library. The website provides a source code zip file as well as some pre-built binaries. For example, under Microsoft Windows 32bit platform, a pre-built OpenCV-2.1.0-win32-vs2008.exe install program is provided. Download and run it on one development machine, where Microsoft visual Studio 2008 (Express Edition is enough) should be installed.

After install, you will find OpenCV dynamic libraries (both release and debug) in C:\OpenCV2.1\bin and corresponding import libraries in C:\OpenCV2.1\lib and a branch of c sample code under C:\OpenCV2.1\samples\c. Some of the sample code files are edge.c(image edge detector), houghlines.c (detect a line in an image), and dft.c(DFT transform).

Step 2. Install the IPP library
Intel IPP is available as a stand-alone product and as a part of the Intel® Parallel Studio 2011, Intel® Parallel Studio XE 2011, Intel® C++ Studio XE2011, Intel® Composer XE 2011, and Intel® C++ Composer XE 2011. Please visit the Intel® Software Evaluation Center to evaluate this product.. One can install any one of these package on the machine where OpenCV were installed.

After install, Intel® IPP dynamic libraries will be located in C:\Program Files\Intel\ComposerXE-2011\redist\ia32\ipp and their corresponding import libraries will be located in C:\Program Files\Intel\ComposerXE-2011\ipp\lib\ia32. Intel® IPP also includes both a serial and a threaded static library in this folder.
The free Intel IPP code samples can be downloaded by going here.

Step 3. Call Intel® IPP functions and call OpenCV functions in the same application.
Take transposing an image as example.
As OpenCV provides friendly-user API and support about 80 image formats, it is easy to use it to read, write and show an image file.
3.1 Load a source image via an OpenCV function: cvLoadImage()
3.2 Create a destination image via an function cvCreateImage();
3.3 Call cvRemap() to transpose the original image or call ippiRemap to transpose the image
Example code:

```
 // Load orig image via OpenCV function
IplImage* pSrcImg = cvLoadImage("testimg.bmp", 1)
 //Create Destination image via OpenCV function
IplImage* pDstImgCV  = cvCreateImage( cvSize(TARGET_WIDTH,TARGET_HEIGHT ),
pSrcImg->depth,pSrcImg->nChannels);
```

`IplImage* pDstImgIPP = cvCreateImage( cvSize(TARGET_WIDTH,TARGET_HEIGHT ), pSrcImg->depth,pSrcImg->nChannels);`	

`// Transpose image via cvRemp`
`cvRemap(pSrcImg, pDstImgCV,`
`    pXMap, pYMap,`
`CV_INTER_LINEAR,`
`cvScalarAll(0));`
	`//Transpose image via ippiRemap`
	`ippiRemap_8u_C3R((Ipp8u*)pSrcImg->imageData,srcSize, pSrcImg->widthStep, roiRect,`
	`      pXTable,  sizeof(float)*TARGET_WIDTH,`
	`pYTable,  sizeof(float)*TARGET_WIDTH,`
	` (Ipp8u*) pDstImgIPP->imageData,`
	`pDstImgIPP->widthStep, dstSize,`
	`       IPPI_INTER_NN);`

For techinical details of cvRemap() or ippiRemap(), please read the reference manual of OpenCV openCV.pdf and IPP reference manual.

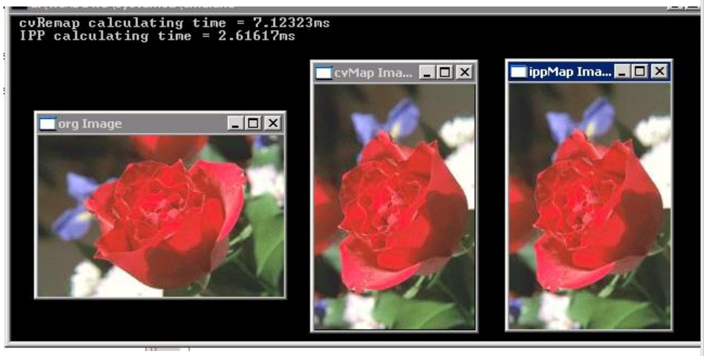
3.4 Link the required libraries to your project

For example, in Microsoft Visual Studio Project Property =>Linker=>input=>Additional Dependencies, add the libraries listed below:

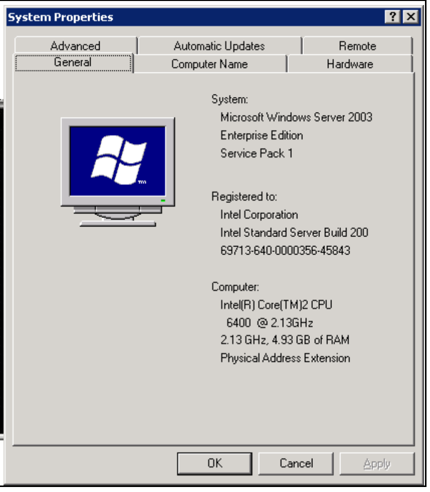
`//typical OpenCV library`	`//typical IPP library`
cv210.lib cxcore210.lib highgui210.lib (Add the include path and library path in Project Property also)	ippi.lib ippcv.lib ipps.lib ippcore.lib (Add the include path and library path in Project Property also)

3.5 Build and run the sample.

```
// Result image and performance
```



```
cvRemap calculating time = 7.12323ms
IPP calculating time = 2.61617ms
```



System Properties

System:
Microsoft Windows Server 2003
Enterprise Edition
Service Pack 1

Registered to:
Intel Corporation
Intel Standard Server Build 200
69713-640-0000356-45843

Computer:
Intel(R) Core(TM)2 CPU
6400 @ 2.13GHz
2.13 GHz, 4.93 GB of RAM
Physical Address Extension

Screenshots of the results are shown above and the run result performance on Intel® Core 2 Duo CPU E6400 is as follows: cvRemp takes 7.123 ms and Intel IPP ippiRemap takes 2.616 ms. The speedup due to Intel IPP is about 2.7x

Our sample uses the same functionality from Intel IPP and OpenCV. Of course a developer can call any different OpenCV or Intel IPP functions depending on their needs. Since IPP is highly optimized on Intel multi-core architecture and OpenCV is widely used in computer vision applications, it is worth adopting both IPP and OpenCV functions in computer vision applications on Intel architecture.

2. Integrate Intel® IPP library into OpenCV library and use OpenCV library with built-in -IPP

2.1 Integrate Intel® IPP library into OpenCV library

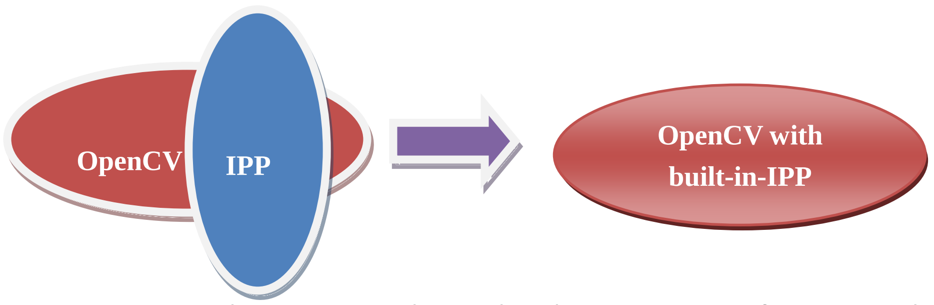
As most OpenCV users noticed, the earlier OpenCV version (i.e OpenCV 1.0) had integrated IPP internally so it could dynamically detect IPP libraries and load them automatically (see Intel® Ipp - Open Source Computer Vision Library (OpenCV) FAQ). But as of OpenCV 2.0, the default builds of OpenCVdo not include IPP. For who still want to use the OpenCV library with IPP, you will need to recompile the OpenCV library.

Developers can build with IPP support as per the instruction given in the OpenCV install guide http://opencv.willowgarage.com/wiki/InstallGuide
Or refer to GUI build instance in IPP Support Model Changed in OpenCV 2.1 (windows).
Below is a command line sample of building IPP library into the OpenCV library under Ubuntu 10.04 64 bit.
Step 1. Download OpenCV source code, which is available from sourceforge.net and extract OpenCV source to the OpenCV 2.1 folder
Step 2. Install IPP, for example, install l_cproc_p_11.1.073 package to get IPP components
Step 3. Download CMAKE and Install it
Step 4. Build with IPP support with command

```
>mkdir OpenCV_IPP_BUILD_61STATIC   (create a result folder for store all build temp file and result library)
>cd OpenCV_IPP_BUILD_61STATIC
>source /opt/intel/Compiler/11.1/073/bin/iccvars.sh intel64
>cmake -D CMAKE_BUILD_TYPE=RELEASE -D USE_IPP=on -D IPP_PATH=/opt/intel/Compiler/11.1/073/ipp/em64t/lib ../
```

After build succeeds, all openCV libraries will be in OpenCV_IPP_BUILD_61STATIC /lib and bin.

The integration processing is like below figure. We call the integrated libraries as OpenCV libraries with built-in-IPP.



Please note OpenCV currently integrates static IPP libraries to dlls by defaults. So using OpenCV DLLs do not need the ipp dll.

2.2 Using the OpenCV library with build-in Intel® IPP.

As we mentioned in IPP Support Model Changed in OpenCV 2.1
OpenCV 2.1.0 integrates Intel® IPP functions when the Macro "HAVE_IPP" is defined during

the process of building cxcore and cv libraries. The built-in IPP functions include

- Color Conversion in cv =>cvcolor.cpp
- Harr classifier training functions in cv => cvharr.cpp
- ippsDFT function in cxcore=>cxdft.cpp

(Please note, the early IPP version included far more IPP functions than 2.1.0, please check the list in __cvipp.h).

DFT(Discrete Fourier Transformation) is widely used in image template match or in image similarity measure. Let's compare two images and find whether the template image is in the source image.

Step 1. Load the source image and the template image via OpenCV function: cvLoadImage()

Step 2. Create destination image via function cvCreateImage();

Step 3. Call OpenCV function `cvMatchTemplate()` which measures image similarity to find the matched image in source image

Example code:

Step 4. Link OpenCV library with built-in Intel® IPP

```
// Load orig image and template via OpenCV function
IplImage *template_image = cvLoadImage ("box.png",0);
IplImage* converted_image= cvLoadImage ("box_in_scene3.png",0);
```

```
 //Create Destination image via OpenCV function
result_ncc = cvCreateImage(
cvSize(converted_image->width -template_image->width+1,
converted_image->height-template_image->height+1),IPL_DEPTH_32F,1);
```

```
/* Measures similarity between template and overlapped windows in the
source image and fills the resultant image with the measurements */
cvMatchTemplate(image_ncc, template_image, result_ncc, CV_TM_CCORR_NORMED);
```

```
 //Link typical OpenCV library with
build_in_IPP. Set the include path and
library path in Project Property:
C:\OpenCV-2.1.0\OpenCV_IPP_BUILD_61STAT
IC\lib and \bin
cv210d.lib cxcore210d.lib
highgui210d.lib
```

```
// Link default OpenCV library
  Set the include path and library
path in Project Property:
C:\OpenCV2.1\lib and \bin

cv210d.lib cxcore210d.lib
highgui210d.lib
```

For techinical details of `cvMatchTemplate()`, please read the reference manual of OpenCV openCV.pdf.

Step 5. Build and run the sample.
In order to compare the result, we also link the default OpenCV library as in the above right column. The link library names are completely same, but the path is different. Remembered that in section 1, default OpenCV library is located in C:\OpenCV2.1\lib and in section 2.1, the OpenCV library with built-in Intel® ipp is located in C:\OpenCV-2.1.0\OpenCV_IPP_BUILD_61STATIC\lib.

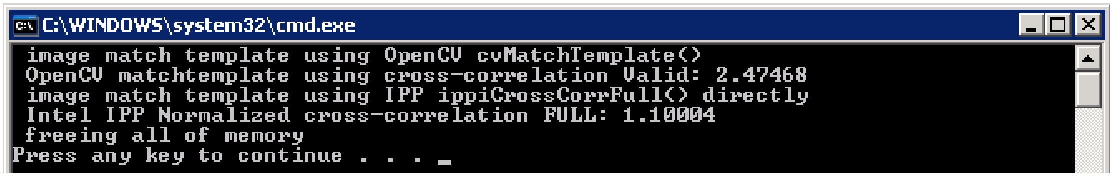
Call the time counter function:
```
t1 = (double)cvGetTickCount();
    for (int j=0;j<10;j++)
     cvMatchTemplate(image_ncc, template_image, result_ncc, V_TM_CCORR_NORMED);
t2 = (double)cvGetTickCount();
timeCalc=(t2-t1)/((double)cvGetTickFrequency()*1000. * 1000.0);
```

Moreover, Intel IPP also includes the same functionality, so we also tried IPP ippiCrossCorrValid_Norm() function in the sample.

Here is the performance data for the operation.

With default OpenCV cvMatchTemplate	with build-in_IPP OpenCV	With IPP ippiCrossCorrValid_Norm()	Build-in/default
5.38s	2.47s	1.1s	2.17x

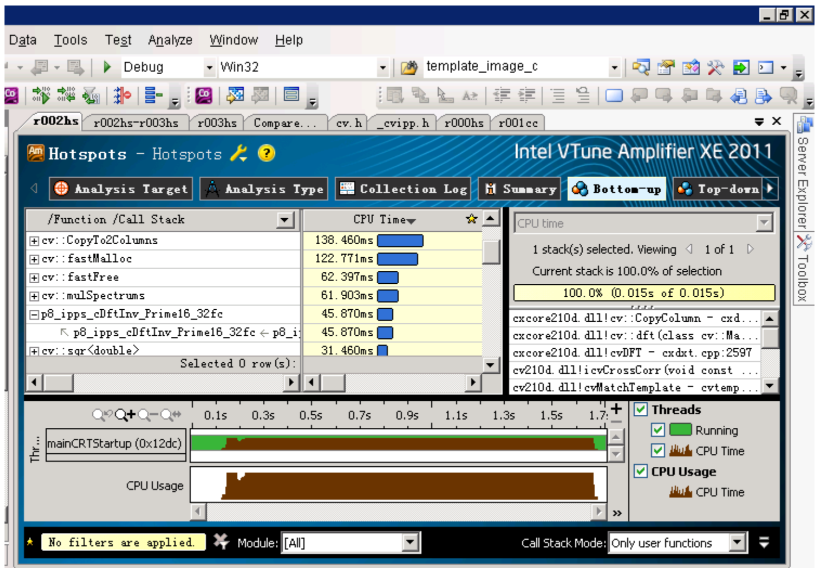
The speed up of built-in_IPP OpenCV vs. default OpenCV is 2.17x. OpenCV with independent Intel IPP is about 2.25x faster than OpenCV with built-in intel® IPP


C:\WINDOWS\system32\cmd.exe
```
 image match template using OpenCV cvMatchTemplate()
 OpenCV matchtemplate using cross-correlation Valid: 2.47468
 image match template using IPP ippiCrossCorrFull() directly
 Intel IPP Normalized cross-correlation FULL: 1.10004
 freeing all of memory
Press any key to continue . . . _
```

3. Performance differences between using OpenCV, OpenCV with built-in Intel® IPP , and Intel® IPP

One may ask why there is a performance gap between them. First, as we know, the default OpenCV function is using genetic code. OpenCV with built-in Intel® IPP uses an IPP function, which is highly optimized for Intel architecture processors. See the screenshot we get via Intel® VTune Amplifier XE 2011.

OpenCV with built-in Intel® IPP will ultimately call a IPP function.
The call graph is cvMatchTemplate() ->icvCrossCorr->cvDFT->...->cv::ippsDFTFwd_CToC->p8_ippsDFTFwd_CToC_32fc.
For more information about "p8" prefix, please see Understanding CPU Dispatching in the Intel® IPP Library



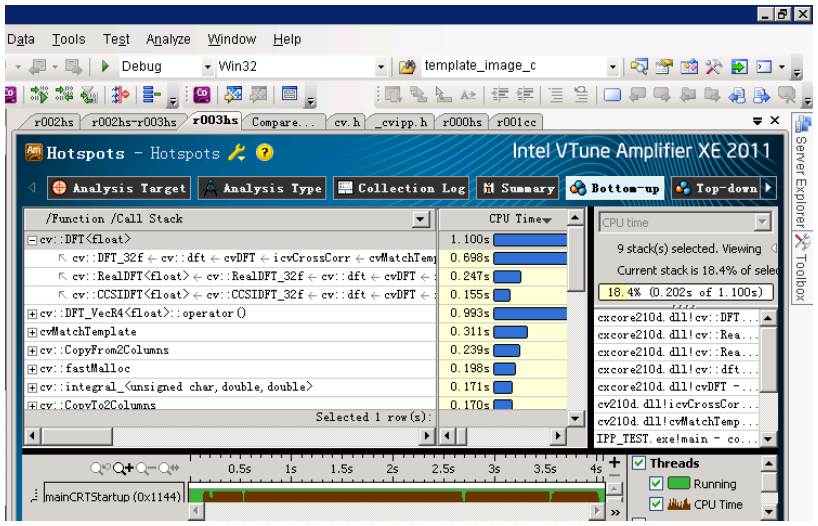
```
r002hs | r002hs-r003hs | r003hs | Compare... | cv.h | _cvipp.h | r000hs | r001cc
Hotspots - Hotspots                                   Intel VTune Amplifier XE 2011

/Function /Call Stack                                     CPU Time
+cv::CopyTo2Columns                                       138.460ms
+cv::fastMalloc                                           122.771ms
+cv::fastFree                                              62.397ms
+cv::mulSpectrums                                          61.903ms
-p8_ipps_cDftInv_Prime16_32fc                              45.870ms
    p8_ipps_cDftInv_Prime16_32fc ← p8_i                    45.870ms
+cv::sqr<double>                                           31.460ms
                              Selected 0 row(s):

CPU time
1 stack(s) selected. Viewing 1 of 1
Current stack is 100.0% of selection
100.0% (0.015s of 0.015s)
cxcore210d.dll!cv::CopyColumn - cxd...
cxcore210d.dll!cv::dft(class cv::Ma...
cxcore210d.dll!cvDFT - cxdxt.cpp:2597
cv210d.dll!icvCrossCorr(void const ...
cv210d.dll!cvMatchTemplate - cvtemp...

mainCRTStartup (0x12dc)
No filters are applied.   Module: [All]   Call Stack Mode: Only user functions
```

With default OpenCV, the call graph is cvMatchTemplate() ->icvCrossCorr->cvDFT->...->cv::RealDft_32f...->cv::DFT, which is written using some generic C code and some SSE2 optimized code.



```
r002hs | r002hs-r003hs | r003hs | Compare... | cv.h | _cvipp.h | r000hs | r001cc
Hotspots - Hotspots                                   Intel VTune Amplifier XE 2011

/Function /Call Stack                                                  CPU Time
-cv::DFT<float>                                                        1.100s
    cv::DFT_32f ← cv::dft ← cvDFT ← icvCrossCorr ← cvMatchTem          0.698s
    cv::RealDFT<float> ← cv::RealDFT_32f ← cv::dft ← cvDFT ←           0.247s
    cv::CCSIDFT<float> ← cv::CCSIDFT_32f ← cv::dft ← cvDFT ←           0.155s
+cv::DFT_VecR4<float>::operator()                                      0.993s
+cvMatchTemplate                                                       0.311s
+cv::CopyFrom2Columns                                                  0.239s
+cv::fastMalloc                                                        0.198s
+cv::integral_<unsigned char, double, double>                          0.171s
+cv::CopyTo2Columns                                                    0.170s
                                                  Selected 1 row(s):

CPU time
9 stack(s) selected. Viewing
Current stack is 18.4% of selec
18.4% (0.202s of 1.100s)
cxcore210d.dll!cv::DFT...
cxcore210d.dll!cv::Rea...
cxcore210d.dll!cv::Rea...
cxcore210d.dll!cv::dft...
cxcore210d.dll!cvDFT -...
cv210d.dll!icvCrossCor...
cv210d.dll!cvMatchTemp...
IPP_TEST.exe!main - co...

mainCRTStartup (0x1144)
```

(Please notes, OpenCV can be built with optimized code, and, starting with v2.0, OpenCV also includes SSE2-optimized code. many of the functions run significantly faster on the modern 32-bit x86 and 64-bit x64 platforms.)

An explanation for the fact that the Intel IPP `ippiCrossCorrValid_Norm()` is faster than OpenCV with built-in Intel® IPP `cvMatchTemplate()`, could be because that except for possibly a slightly different algorithm, OpenCV with build-in Intel® IPP is using the static **serial** IPP library. The intel® IPP function is using the **dynamic** library, in which the function is threaded. When run on multi-core processors, IPP functions will run in parallel.

The comparisons between general OpenCV and Built-in Intel IPP OpenCV demonstrates that the OpenCV library with built-in Intel® IPP does not require an external Intel IPP library. It can be used the same as the standard OpenCV library but it also takes advantage of the optimization of built-in Intel® IPP functions. However, if comparing IPP with build-in Intel IPP OpenCV (The precondition is that there is similar functionality in IPP), using IPP directly is flexible and may produce better performance due to internal threading.

Please note, when use OpenCV library with build-in Intel® IPP and IPP library together, we recommend using the same version and dynamic version of Intel IPP library if possible in order to avoid potential conflicts. This requires modify the IPP link configuration file in OpenCV source code.

Summary

Intel® IPP is a software library provided by Intel® software group. It offers over 10K highly optimized fundamental functions for multimedia, data and image processing application. OpenCV is free and open for both non-commercial and commercial use. It provides cross-platform middle-to-high level functions for popular computer vision algorithms. OpenCV can be built with Intel® Integrated Performance Primitives (IPP). This makes it fast on all the architectures supported by the library, where the optimal code for each host architecture is chosen at runtime.

In this paper, we showed how to integrate Intel® IPP into OpenCV step by step. With the two samples, we also discovered how OpenCV library with built-in Intel® IPP works, demonstrated how to use Intel® IPP with latest OpenCV library as well as how to use OpenCV with built-in Intel® IPP. In summary, the OpenCV library supports more computer vision algorithms than the Intel IPP library. Intel IPP library provides low-level but high performance basic image and video processing functions. We can use them as two independent libraries. However, it is convenient to integrate IPP and OpenCV via OpenCV build process and the Have_IPP flag. Thus we can benefit Intel IPP integrated into the OpenCV library automatically. In regard to the performance of routines with the same functionality in Intel® IPP and OpenCV, the direct IPP function call has the best performance. It is about 5x faster than the default OpenCV library and 2.21x the OpenCV library with built-in ipp under two cores @2.13G. Taking considering of the fact of OpenCV itself can be built in parallel and OpenCV also includes SSE2-optimized code for many of functions, we recommend using OpenCV with calls to stand alone Intel® IPP, or OpenCV with built-in Intel IPP according to the application's feature and performance requirements.

Refer to our Optimization Notice for more information regarding performance and optimization choices in Intel software products.