

CASE STUDY



Optimizing an Illegal Image Filter System

Intel® Integrated Performance Primitives

High-Performance Computing



Tencent Doubles the Speed of its Illegal Image Filter System using SIMD Instruction Set and Intel® Integrated Performance Primitives

For a large Internet service provider like China's Tencent, being able to detect illegal images is important. Popular apps like WeChat*, QQ*, and QQ Album* are not just text oriented, but also image generation and sharing apps. Every year, the volume of newly generated images reach about 100 petabytes—even after image compression. Some users may try to upload illegal images (e.g., porn). They certainly don't tell the system that the image is illegal, so the system runs a check on each image to try to block them. This is a huge computing workload, with billions of images uploaded each day.

Technical Background

When a user uploads an image, the filter system decodes the image for preprocessing. The preprocessing uses a Filter2D function to smooth and resize an image of any size to one fixed size. This intermediate image is the input for fingerprint creation. The new fingerprint is compared to the seed fingerprint of an illegal image. If the image is illegal, it is blocked or deleted and the fingerprint is added into the illegal image fingerprint repository.

Figure 1 shows how a fingerprint seed is added to the repository manually.

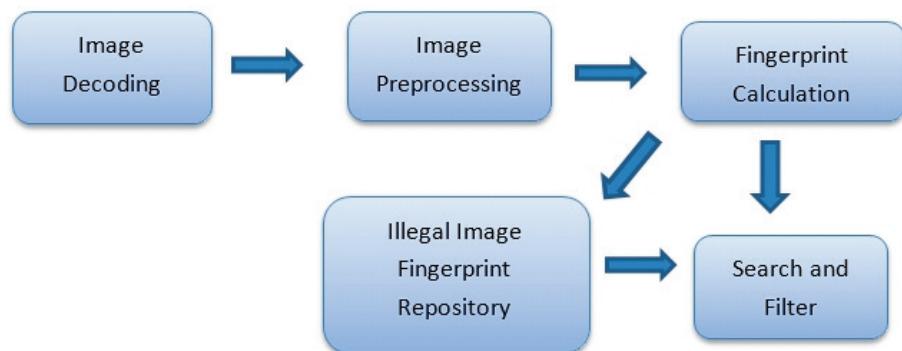


Figure 1. Illegal image filter system architecture

Tencent More than Doubles the Speed and Performance of Filtering

Optimization Overview

Working with Tencent engineers, Intel prepared the environment to tune the hotspot functions. Intel® VTune™ Amplifier is a powerful tool that can provide

runtime information on code performance for developers creating serial and multithreaded applications. With this profiler information, a developer can analyze the algorithm choices and identify where

and how the application can benefit from available hardware resources.

Using Intel VTune Amplifier to profile Tencent's illegal image filter system, the team was able to locate the hotspot functions. Figure 2 shows that GetRegionID, GetFingerGrayRegionHistogram and cv::Filter2D are the top three hotspot functions.

The team found that GetRegionID was called by GenFingerGrayRegionHistogram. The GetRegionID function included a cascaded clause (if else), which is hard to optimize. The GetFingerGrayRegionHistogram function can be reimplemented using SIMD instructions. And the filter2D function can be reimplemented by Intel® Integrated Performance Primitives (Intel® IPP). Both these functions achieved more than a 10x speed-up. The whole system has more than doubled its speed, which helped Tencent double the performance of its system. Moreover, the latency reduction improves the usage experience when users share images and send them to friends. Since Filter2D is also widely used in data processing and digital image processing, Intel's work with Tencent is a good example for other cloud service users.

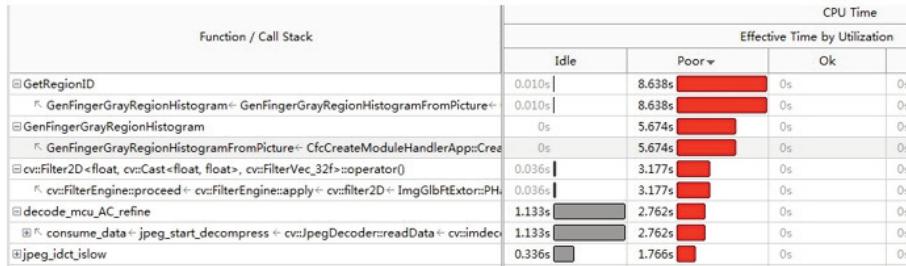


Figure 2. Illegal image filter system hotspot functions

Original Version Simhash.cpp	Optimized Version Using SIMD (simhash_11.cpp)
<pre>static std::string GenFingerGrayRegionHistogram(const cv::Mat& grayMat) { int histogram[9][4] = {0}; ... for(int h = 0; h < grayMat.rows; h++) { for(int w = 0; w < grayMat.cols; w++) { unsigned char v = grayMat.at<unsigned char>(h, w); int iRegionID = GetRegionID(w, h, grayMat.cols, grayMat.rows, 3); if(iRegionID < 0 iRegionID >= 9) continue; ... } } static int GetRegionID(int cw, int ch, int w, int h, int div, int skipDiv = 0/*skip boundary*/) { int divWidth = w/div; if(0 == divWidth) divWidth = 1; int wid = cw/divWidth; if(wid >= div) wid = div - 1; int divHeight = h/div; if(0 == divHeight) divHeight = 1; int hid = ch/divHeight; if(hid >= div) hid = div - 1; if(wid < skipDiv wid > div - 1 - skipDiv hid < skipDiv hid > div - 1 - skipDiv) return -1; ... } }</pre>	<pre>static std::string GenFingerGrayRegionHistogram(const cv::Mat& grayMat) { int histogram[9][4] = {0}; ... for(int h = 0; h < grayMat.rows; h++) { unsigned char const *p = grayMat.ptr<unsigned char>(h); int nextRegionCw = 0; int regionSize; /*TODO: require more consideration for 3 in *grayMat.cols/3*3 if *• the region block schematic changed */ for(int w = 0; w < grayMat.cols; w+=regionSize) { regionSize = GetNextRegionIDSize(w,h,grayMat.cols, grayMat.rows, 3, &iRegionID); ... for(int i=0;i<(regionSize&(~0x1));i+=16) { /* The region iRegionID fixed within this loop*/ __m128i xmm0; //tmp xmm0 = _mm_loadu_si128((__m128i*)(p+i+w)); ... xmm1 = _mm_srli_epi16(xmm0, 6); xmm1 = _mm_and_si128(xmm1,xmm15); xmm2 = _mm_and_si128(xmm0,xmm14); ... } } } }</pre>

Figure 3. Before and after optimization

Intel® Streaming SIMD Extensions and GetFingerGrayRegionHistogram Optimization

Intel introduced an instruction set extension with the Intel® Pentium® III processor called Intel® Streaming SIMD Extensions (Intel® SSE). This was a major extension that added floating-point operations over the earlier, integer-only SIMD instruction set called MMX™, which was introduced with the Intel Pentium processor. Since the original Intel SSE, SIMD instruction sets have been extended by wider vectors, new and extensible syntax, and rich functionality. The latest SIMD instruction, set Intel® Advanced Vector Extensions (Intel® AVX512), can be found in the Intel® Core™ i7 processor.

Algorithms that significantly benefit from SIMD instructions include a wide range of applications such as image and audio/video processing, data transformation and compression, financial analytics, and 3D modeling and analysis.

The hardware that runs Tencent's illegal image filter system uses a mix of processors. Intel's optimization is based the common SSE2, which is supported by all the Tencent systems.

The team investigated the top two functions: GetRegionID and GetFingerGrayGegionHistogram. GetRegionID was called by GenFingerGrayRegionHistogram. The GetRegionID function includes a cascaded clause of "if else," which is hard to optimize. But the function GetFingerGrayGegionHistogram has a big "for" loop for each image pixel, which has room for optimization. And there is a special algorithm logical when GetFingerGrayGegionHistogram calls GetRegionID. Intel rewrote part of the

code to use the SIMD instruction to optimize the GetFingerGrayGegionHistogram function. Figure 3 shows Simhash.cpp, which is the source file before optimization, and simhash11.cpp, the file after optimization.

Intel tested the optimized code on an Intel® Xeon® processor E5-2680 0 at 2.70GHz. The input image was a large JPEG image (5,000 by 4,000 pixels). Tencent calls the function by single thread, so the team tested it on single core. The result was a significant speedup (Table 1).

Intel® IPP and Filters 2D Optimization

Intel SIMD capabilities change over time, with wider vectors, newer and more extensible syntax, and rich functionality. To reduce the effort needed on instruction-level optimization to support every processor release, Intel offers a storehouse of optimized code for all kinds of special computation tasks. The Intel IPP library is one of them. Intel IPP provides

developers with ready-to-use, processor-optimized functions to accelerate image, signal, data processing, and cryptography computation tasks. It supplies rich functions for image processing, including an image filter, which was highly optimized using SIMD instructions.

Optimizing Filters 2D

Filtering is a common image processing operation for edge detection, blurring, noise removal, and feature detection. General linear filters use a general rectangular kernel to filter an image. The kernel is a matrix of signed integers or single-precision real values. For each input pixel, the kernel is placed on the image in such a way that the fixed anchor cell (usually be geometric center) within the kernel coincides with the input pixel. Then it computes the accumulation of the kernel and the corresponding pixel. Intel IPP supports the variant filter operation (Table 2).

For several border types (e.g., constant border, replicated border), Tencent used OpenCV* 2.4.10 to preprocess images. Using Intel VTune Amplifier XE, the team found that the No. 3 hotspot is cv::Filter2D. It was convenient and easy to use the Intel IPP filter function to replace the OpenCV function.

The test file filter2D_demo.cpp is from OpenCV Version: 3.1:

- @brief Sample code that shows how to implement your own linear filters by using filter2D function
- @author OpenCV team

The team used the IPP filter function from Intel IPP 9.0, update 1, as shown in Figure 4.

Accelerating the Filter2D by Intel IPP

The team used the same filter kernel to filter the original image, and then recompiled the test code and OpenCV code with GCC and benchmarked the result on Intel Xeon processor-based systems. The effects of the image filter are the same from Intel IPP code and OpenCV code.

Table 1. Traits and behaviors of mutexes

Algorithm Name	Latency (Seconds) (Smaller is Better)	Gain
Tencent's original	0.457409	100%
SSE4.2 optimized	0.016346	2,798%

Table 2. Variant filter operation

IPP Filter Functionality	Notes
ppiFilter<Bilateral Box SumWindow>Border	Perform bilateral, box filter, sum pixel in window
ippiMedian Filter	Median filter
ippiGeneral Linear Filters	A user-specified filter
Separable Filters	FilterColumn or FilterRow function
Wiener Filters	Wiener filter
Fixed Filters	Gaussian, laplace, hipass and lowpass filter, Prewitt, Roberts and Scharr filter
Convolution	

Figure 5 shows the Intel IPP smooth image (right) compared to original image.

The consumption times (Table 3) show that the `ipp_filter2D` take less time (about 9ms), while the OpenCV source code takes about 143ms. The IPP filter2D is 15x faster than the OpenCV plain code.

Note that OpenCV has Intel IPP integrated during compilation; thus, some functions can call Intel IPP functions underlying automatically. In some of OpenCV versions, the `Filter2d` function can at least call the IPP Filter2D. But, considering the overhead of function calls, the team selected to call the IPP function directly.

Summary

With manual modifications to the algorithm, plus SSE instruction optimization and Intel IPP filter replacement, Tencent reports that the performance of its illegal image filter system has more than doubled compared to the previous implementation in its test machine. Figure 6 shows the total results (category 3).

Tencent is a leading Internet service provider with high performance requirements for all kinds of computing tasks. Detecting and filtering illegal images is a typical example. Popular apps like Wechat, QQ, and QQ Album generate billions of images each day.

Intel helped Tencent optimize the top three hotspot functions of its illegal image filter system. By manually implementing the fingerprint generation with SIMD instructions, and replacing the `filter2D` function with a call into the Intel IPP library, Tencent was able to speed up both hotspots by more than 10x. As a result, the entire illegal image filter system has more than doubled its speed. This will help Tencent double the capability of its systems. Moreover, the latency reduction improves the user experience for customers sharing and sending images.

filter2D OpenCV code	IPP Filter2D
<pre>filter2D(src, dst, ddepth, kernel, anchor, delta, BORDER_REPLICATE); /// Initialize arguments for the filter anchor = Point(-1, -1); ... /// Update kernel size for a normalized box filter kernel_size = 3 + 2*(ind%5); // 3, 5, 7, 9, 11 kernel = Mat::ones(kernel_size, kernel_size, CV_32F) / (float)(kernel_size*kernel_size); struct timeval start, finish; gettimeofday(&start, NULL); /// Apply filter filter2D(src, dst, ddepth, kernel, anchor, delta, BORDER_REPLICATE); gettimeofday(&finish, NULL); double duration_1=(finish.tv_sec-start.tv_sec)*1000+(finish.tv_usec-start.tv_usec)/1000; printf("opencv cost time: %f ms\n", duration_1); imwrite("./opencv_output.jpg", dst); }</pre>	<pre>ipp_filter2D(src, dst, ddepth, kernel, anchor, delta, BORDER_REPLICATE); typedef IppStatus (CV_STDCALL *ippiFilterBorder)(const void * pSrc, int srcStep, void *pDst, int dstStep, IppiSize dstRoiSize, IppiBorderType border, const void *borderValue, const IppiFilterBorderSpec* pSpec, Ipp8u* pBuffer); int stype = src.type(), sdepth = CV_MAT_DEPTH(stype), cn = CV_MAT_CN(stype), ktype = kernel.type(), kdepth = CV_MAT_DEPTH(ktype); Point ippAnchor((kernel.cols-1)/2, (kernel.rows-1)/2); int borderTypeNI = borderType & ~BORDER_ISOLATED; IppiBorderType ippBorderType = ippiGetBorderType(borderTypeNI); /* convert to IPP BorderType */ /* find Filter */ ippiFilterBorder ippFunc = stype == CV_8UC1 ? (ippiFilterBorder)ippiFilterBorder_8u_C1R : ... ippiFilterBorderInit_32f(pKerBuffer, kernelSize, dataType, cn, ippRndFinancial, spec); status = ippFunc(src.data, (int)src.step, dst.data, (int)dst.step, dstRoiSize, ippBorderType, borderValue, spec, pBuffer);</pre>

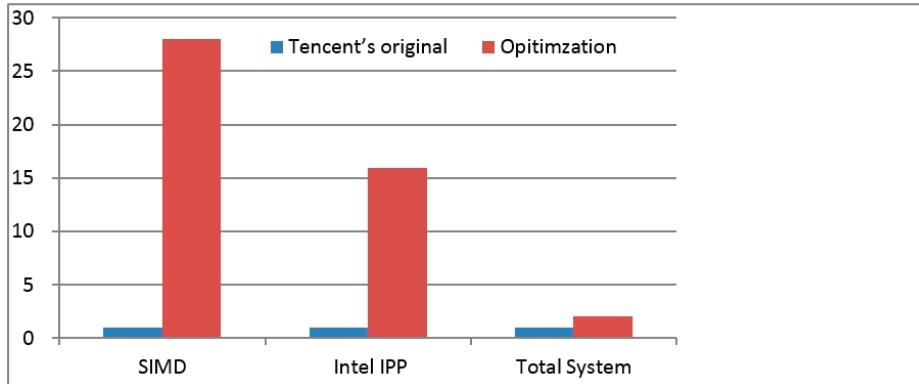
Figure 4. Code before (left) and after (right) using IPP



Figure 5. Image before (left) and after filtering

Table3. Consumption times

CPU	Program	Time (ms)
Intel® Xeon® processor E5-2680 v3	<code>ipp_filter2D</code>	9
	<code>OpenCV_filter2D</code>	143

[Learn More](#)[Tencent >](#)[Intel Integrated Performance Primitives >](#)

Configuration Info - Versions: Intel® IPP 9.0.1; Hardware: Intel® Xeon CPU E5-2680 2.7GHz, AVX Supported, 8MB Cache, 8 GB Memory; Operating System: RHEL 6.3 GA x86_64; Total system test Source: Tencent Corporation on Intel(R) Xeon(R) CPU E3-1230 V2, GCC 4.4.6, Linux: 2.6.32.43
 Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, refer to www.intel.com/performance/resources/benchmark_limitations.htm.
 • Other brands and names are the property of their respective owners.
 Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

Figure 6. Total results



Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software, or service activation.

Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer, or learn more at www.intel.com.

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to www.intel.com/performance.

Intel does not control or audit the design or implementation of third party benchmark data or Web sites referenced in this document. Intel encourages all of its customers to visit the referenced Web sites or others where similar performance benchmark data are reported and confirm whether the referenced benchmark data are accurate and reflect performance of systems available for purchase.

This document and the information given are for the convenience of Intel's customer base and are provided "AS IS" WITH NO WARRANTIES WHATSOEVER, EXPRESS OR IMPLIED, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS. Receipt or possession of this document does not grant any license to any of the intellectual property described, displayed, or contained herein. Intel® products are not intended for use in medical, lifesaving, life-sustaining, critical control, or safety systems, or in nuclear facility applications.

Copyright © 2016 Intel Corporation. All rights reserved. Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

* Other names and brands may be claimed as the property of others.

Printed in USA

0516/SS

Please Recycle