White Paper: Media Cloud – Video Processing Solutions with Intel® Platform Technology

Contents

Contents ................................................................................................................................................. 3
1. Abstract ........................................................................................................................................... 4
2. Media Cloud Introduction .................................................................................................................. 4
   2.1 Offline Video Transcoding Scenarios ......................................................................................... 5
   2.2 Real-Time Video Transcoding Scenarios ................................................................................. 6
3. Intel® Xeon™ Platform Solutions .................................................................................................... 7
   3.1 CPU based Software Transcoding Solution ............................................................................. 7
   3.2 GPU based Hardware Transcoding Solution .......................................................................... 21
4. Industry Deployment and Direction ................................................................................................. 28
   4.1 Distributed Processing Platform ........................................................................................... 29
   4.2 HEVC/H.265 Codec ............................................................................................................... 29
   4.3 Data Mining and Others ...................................................................................................... 30
5. Performance Tuning Methodology in Media Cloud Domain .......................................................... 30
   5.1 System Level Tuning ............................................................................................................ 31
   5.2 Application Level Tuning .................................................................................................... 34
   5.3 Micro-Arch Level Tuning .................................................................................................... 36
6. Summary .......................................................................................................................................... 38
Reference .............................................................................................................................................. 38

Author contacts
Yang Lu, Senior Application Engineer <yang.y.lu@intel.com>
1. Abstract

Media processing is one of the hottest applications in the IPDC field especially for video and image processing which consumes lots of computing resources as well as storage. Media Service Providers have been devoted to improving the computing servers’ efficiency, especially for the video transcoding, analysis, and searching, streaming related workloads. In this paper, we analyze the architecture and platforms for the most popular media applications, figure out the performance bottleneck and propose technical solutions based on both of Intel® CPU and GPU platforms. We further investigate the media cloud trend and direction, illustrate in details how to take full advantage of Intel® Architecture® platform’s capabilities to achieve excellent performance.

2. Media Cloud Introduction

Media cloud applications and businesses have been increasing significantly in recent years driven by mobile internet, Social network and entertainment demands, such as video on demand (VOD) service and online video sharing, IPTV and broadcasting. As mobile tablet and smart phones have been more and more popular, users can share their videos and watch TV/movies whenever and wherever possible. In the enterprise business, remote education and communication, surveillance are growing rapidly, too. Video accounts for 70% of all internet traffic today \( \text{(Source: Sandvine Global Internet Phenomena Report, 2H 2013)} \) , and more and more media cloud applications emerging and population in recent years, huge volume of videos are generated and uploaded/downloaded every day, how to process them in most efficient way is a big challenge for those media Service providers. In China, more than 20 media ISVs are pursuing the most efficient transcoding solution and platform to achieve best performance and maintain the video quality at the same time for better user experience as well as lower TCO.

Generally, the major media cloud usage modules are:

- **Video Delivery Service**: deliver the video content to different devices any time when it’s requested. The video sources include the user generated content, the TV programs, movies, and other device generated content.
- **Video Communication**: video broadcasting and video conference businesses have been growing rapidly these years.
- **Video Surveillance**: video camera captures huge volume data every day, which will be transferred to the server for further processing and analyzing.
- **Video Cloud and Cloud Based Display/Gaming**: like the virtual hosted desktop, cloud gaming and workstation management in cloud. Lots of video data need to be processed at server side to support client applications via network.
- **Video Analysis and Big Data**: video indexing and searching, object recognition, further data mining from the content, automated advertisement promotion and new business analyze.
In these media related applications, the basic processing modules are video transcoding, editing, feature extraction and analyze. Furthermore video transcoding is the primary workload that consumes the most of computing resource, and becomes the fundamental function for other further processing. In current media cloud platforms, there are two major types of video transcoding scenarios: offline video transcoding and real-time video transcoding. They have some different business requirements and technical focus.

In this paper, we will introduce current media cloud applications architecture and technical challenge, then investigate how to take advantage of the Intel® Architecture® platform to meet technical challenges and achieve the excellent performance.

### 2.1 Offline Video Transcoding Scenarios

Media Service Providers need to trans-code all the stored videos, either self-generated, licensed content or user generated & uploaded content, to the specific format in order to facilitate the management and maintain better TCO. Generally, as shown in the figure 1, there are three primary modules in the offline video transcoding applications:

1) Transcode all the media files to the target format, including necessary video editing and scaling.
2) Distribute the trans-coded files to the distributed storage nodes for user to download in the most effective way.
3) Deliver the content to user that they preferred.

![Figure 1: offline video transcoding application architecture](image)
In the offline transcoding clusters, all kind of the video files need to be trans-coded to the selected format, like mpeg2, H.264, or others. During the transcoding process, items below should be noticed:

1. Video scaling: video files will be scaled to the different resolutions to meet the client device decoding requirement, which would help achieve best user experience and save network bandwidth.
2. Video editing: during these process some necessary editing, such as adding watermark, logo, and subtitles will be done
3. Video quality control: as the HD (High Definition) and SHD (Super High Definition) videos are getting more and more popular, video quality is also a critical performance measurement for the transcoding solutions.
   a) Lots of enhancement functions have been adopted to improve the video quality, such as de-interlacing, de-noising, anti-blurring, post-soften, post-smoothing etc.
   b) Two-pass transcoding: 2-pass transcoding mechanism is deployed already. Two pass encoding, also known as the multi-pass encoding, is a video encoding strategy used to retain the best quality during the conversion. In the first pass, video encoder analyzes the video from the beginning to the end and save the data to log file. In the second pass, the encoder scans and writes video according to the log file, determining the possible way to maintain the best quality within the video bitrate limit. Therefore 2-pass encoding is only used in VBR (Variable bitrate) scenario, the CBR (constant bitrate) encoding doesn't offer any flexibility for the encoder to determine the bitrate for each frame. The computing complexity of the 2-pass encoding is far more than two times than the one pass encoding, which demands much more computing resource and consumes longer time.

Video transcoding and all above processes are typical CPU and memory intensive workload, whose performance highly depends on the transcoding system’s computing capabilities. Transcoding clusters’ throughput is the major performance metric in these applications.

Trans-coded videos will be uploaded to the distributed storage clusters to facilitate user downloading or streaming out in the fastest way. CDN (Content Delivery Network) is the most common architecture for now, which deliver large distributed content in multi datacenters across the internet. I/O throughput and availability are the key factors to measure the performance.

2.2 Real-Time Video Transcoding Scenarios

In the real-time video applications, such as interactive video communication, IPTV (Internet Protocol Television) and video broadcasting, users browse the video content real-time and input their new data or control information to the backend server, the encoding cluster generate next round of video stream according to the demand. As shown in Figure 2, these scenarios raise demanding requirement to the latency of the encoding
server, video quality may not be so critical but keeping the video stream stable and smooth.

For the latency intensive workload, GPU (graphics processing unit) based hardware video processing solution and specific video card have been developed these years. Their highly parallel structure and the fixed acceleration unit in hardware make the video processing more effective in terms of the single task calculating speed. From technical perspective, the programming flexibility should be considered in hardware solution.

3. Intel® Xeon™ Platform Solutions

To address those media processing challenge and provide efficient media cloud solutions, Intel develops series of server platforms and related technologies. In this section, we will introduce these platform technologies for those two typical applications that we illustrate in the section 2.

3.1 CPU based Software Transcoding Solution

Offline Video transcoding application could be CPU and memory intensive workload, which requires higher capabilities of the server platform, such as computing efficiency, reliability, and stability. To analyze the relationship of CPU capabilities and transcoding
performance, we take three video transcoding tasks (1080p, 720p and 480p) running on Intel® Xeon™ processor based Platform, with 3 different configurations as shown in the Table 1.

<table>
<thead>
<tr>
<th>Intel® Xeon™ Platform</th>
<th>Processor Number</th>
<th># of Cores</th>
<th># of Threads</th>
<th>Clock Speed</th>
<th>Max Turbo Frequency</th>
<th>Intel® Smart Cache</th>
<th>Intel® QPI Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>E5-2620V2</td>
<td>6</td>
<td>12</td>
<td>15 MB</td>
<td>7.2 GT/s</td>
<td>2.1 GHz</td>
<td>15 MB</td>
<td>7.2 GT/s</td>
</tr>
<tr>
<td>E5-2640V2</td>
<td>8</td>
<td>16</td>
<td>20 MB</td>
<td>7.2 GT/s</td>
<td>2.0 GHz</td>
<td>20 MB</td>
<td>7.2 GT/s</td>
</tr>
<tr>
<td>E5-2697V2</td>
<td>12</td>
<td>24</td>
<td>30 MB</td>
<td>8 GT/s</td>
<td>2.7 GHz</td>
<td>30 MB</td>
<td>8 GT/s</td>
</tr>
</tbody>
</table>

Table 1: Intel® Xeon™ Ivy Bridge based Platform Configuration

We use the x.264 v0.142 and ffmpeg v2.1.4, the open source software video transcoding codec and framework to execute the tasks. The performance metric is the fps (frame per second), higher is better. From Figure 3, we can see the transcoding throughput grows along with CPU frequency, cache size and the number of the cores increasing. Therefore, the high-bin processor is more efficient for video transcoding applications.

For those typical CPU bound applications, IA (Intel® Architecture) platforms offer several advanced the technologies to boost the performance, such as SIMD (single instruction multi data) vectorization, Simultaneous Multithreading (SMT) and serials of high performance libraries.
3.1.1 SIMD Vectorization for Media Processing

Most of the time-consuming video and image processing functions are the block based data intensive computing, which can be optimized by the IA SIMD vectorization instructions. SIMD instructions process multiple data sets within one single CPU cycle, which will greatly improve the data throughput and execution efficiency. SIMD have been widely supported at x86 processors, evolving from MMX, SSE, AVX, to the AVX2 at different x86 platform generations.

We take three videos (1080p, 720p and 480p) for examples, running transcoding tasks on the Intel® Xeon™ E5-2697 v2 (Ivy Bridge) processor based server system. In figure 4, we can see the original transcoding codec’s hot functions, such as the block based filtering, quantization, and DCT related computing could be optimized by SIMD from.

![Figure 4: Hot Functions at One Video Transcoding Application](image)

After enabling the SIMD instructions in this application, as demonstrated in the figure 5, IA SIMD (SSE/AVX) contribute 2-3x performance improvement on both of latency and throughput.
We can observe from the figure 6, many hot functions have been optimized by the SIMD instructions.
There are three methods to use the IA SIMD technology:

1) Compiler auto vectorization:
   Currently almost all the compilers support the IA SIMD auto vectoring, such as gcc, icc[5] (Intel® Compiler), and Microsoft compiler. Launch the compiler with the different vectoring switches when you want to enable the SIMD.

2) High performance vectoring library:
   Intel has developed full set of high performance library[4], use them to replace your original non-optimized API to complete the same functionality but achieve better performance. We will demonstrate one case that utilize the IPP[6] (Intel® Performance Primitive) to optimize the video scaling application in the following section. Furthermore, more and more open source libraries have integrated IA SIMD optimized code, such as x.264, x.265 and ffmpeg, customer can take advantage of IA platform easily and conveniently.

3) Writing SIMD instruction code:
   In some cases, the logic of the code can’t be optimized by the compiler automatically, and there’s no similar high performance API can be adopted directly, we need to analyze the code and rewrite them with SIMD instructions manually according to the programmer guide and related documents[7]. If the scalar block computing code can be rewrite with SIMD fully, even with the assembly tuning, a very good performance boost can be expected. We take a common 64*64 block computing in video/image processing as an example here to demonstrate how to utilize the SSE and AVX2 intrinsic to optimize the original code:

**Code example for 64*64 block computing**

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "smmintrin.h"
#include "immintrin.h"

/********** original block computing serial scalar computing **********/
#define PIXEL_SAD_C( func_type, name, lx, ly )
func_type int name( pixel *pix1, int i_stride_pix1,pixel *pix2, int i_stride_pix2 )
{
    int sum = 0;
    int x, y;
    for( y = 0; y < ly; y+=2 )
    {
        for( x = 0; x < lx; x++ )
        {
            sum += abs( pix1[x] - pix2[x] );
        }
        pix1 += i_stride_pix1<<1;
        pix2 += i_stride_pix2<<1;
    }
    return sum << 1;
}
PIXEL_SAD_C( static, LENT_sad_64x64_c, 64, 64 )

#define SAD4( w, h )
static void LENT_sad4_##w##x##h##_c( pixel *fenc, pixel *p0, pixel *p1, pixel *p2, pixel *p3, int i_stride, int cost[4] )
{
    cost[0] = LENT_sad_##w##x##h##_c( fenc, FENC_STRIDE, p0, i_stride );
    cost[1] = LENT_sad_##w##x##h##_c( fenc, FENC_STRIDE, p1, i_stride );
    cost[2] = LENT_sad_##w##x##h##_c( fenc, FENC_STRIDE, p2, i_stride );
    cost[3] = LENT_sad_##w##x##h##_c( fenc, FENC_STRIDE, p3, i_stride );
}
```
cost[3] = LENT_sad_##w##x##h##.c( fenc, FENC_STRIDE, p3, i_stride );
}

SAD4( 64, 64 )
SAD4( 32, 32 )

******* SSE instruction implementation **************
void inline sad4_64_fast_sse( pixel *fenc, pixel *p0, pixel *p1, pixel *p2, pixel *p3, int i_stride, int cost[4], int ly )
{
    __m128i sum = _mm_setzero_si128();
    int i;

    i_stride <<= 1;
    for( i = 0; i < ly; i += 2 )
    {
        __m128i se = _mm_load_si128( (__m128i *)(fenc) );
        __m128i s0 = _mm_loadu_si128( (__m128i *)(p0) );
        __m128i s1 = _mm_loadu_si128( (__m128i *)(p1) );
        __m128i s2 = _mm_loadu_si128( (__m128i *)(p2) );
        __m128i s3 = _mm_loadu_si128( (__m128i *)(p3) );

        s0 = _mm_sad_epu8( se, s0 );
        s1 = _mm_sad_epu8( se, s1 );
        s2 = _mm_sad_epu8( se, s2 );
        s3 = _mm_sad_epu8( se, s3 );

        s0 = _mm_hadd_epi32( s0, s1 );
        s1 = _mm_hadd_epi32( s2, s3 );
        sum = _mm_add_epi32( sum, _mm_hadd_epi32( s0, s1 ) );

        se = _mm_load_si128( (__m128i *)(fenc + 16) );
        s0 = _mm_loadu_si128( (__m128i *)(p0 + 16) );
        s1 = _mm_loadu_si128( (__m128i *)(p1 + 16) );
        s2 = _mm_loadu_si128( (__m128i *)(p2 + 16) );
        s3 = _mm_loadu_si128( (__m128i *)(p3 + 16) );

        s0 = _mm_sad_epu8( se, s0 );
        s1 = _mm_sad_epu8( se, s1 );
        s2 = _mm_sad_epu8( se, s2 );
        s3 = _mm_sad_epu8( se, s3 );

        s0 = _mm_hadd_epi32( s0, s1 );
        s1 = _mm_hadd_epi32( s2, s3 );
        sum = _mm_add_epi32( sum, _mm_hadd_epi32( s0, s1 ) );

        se = _mm_load_si128( (__m128i *)(fenc + 32) );
        s0 = _mm_loadu_si128( (__m128i *)(p0 + 32) );
        s1 = _mm_loadu_si128( (__m128i *)(p1 + 32) );
        s2 = _mm_loadu_si128( (__m128i *)(p2 + 32) );
        s3 = _mm_loadu_si128( (__m128i *)(p3 + 32) );

        s0 = _mm_sad_epu8( se, s0 );
        s1 = _mm_sad_epu8( se, s1 );
        s2 = _mm_sad_epu8( se, s2 );
        s3 = _mm_sad_epu8( se, s3 );

        s0 = _mm_hadd_epi32( s0, s1 );
        s1 = _mm_hadd_epi32( s2, s3 );
        sum = _mm_add_epi32( sum, _mm_hadd_epi32( s0, s1 ) );
    }
}
sum = _mm_add_epi32( sum, _mm_hadd_epi32( s0, s1 ) );

se = _mm_load_si128( (__m128i *)(fenc + 48) );
s0 = _mm_loadu_si128( (__m128i *)(p0 + 48) );
s1 = _mm_loadu_si128( (__m128i *)(p1 + 48) );
s2 = _mm_loadu_si128( (__m128i *)(p2 + 48) );
s3 = _mm_loadu_si128( (__m128i *)(p3 + 48) );

s0 = _mm_sad_epu8( se, s0 );
s1 = _mm_sad_epu8( se, s1 );
s2 = _mm_sad_epu8( se, s2 );
s3 = _mm_sad_epu8( se, s3 );

s0 = _mm_hadd_epi32( s0, s1 );
s1 = _mm_hadd_epi32( s2, s3 );
sum = _mm_add_epi32( sum, _mm_hadd_epi32( s0, s1 ) );

fenc += (2*FENC_STRIDE);
p0 += i_stride;
p1 += i_stride;
p2 += i_stride;
p3 += i_stride;

} _mm_storeu_si128( (__m128i *)cost, _mm_slli_epi32( sum, 1) );
} }

/****** AVX2 instruction implementation *************/
void inline sad4_64_fast_avx2(pixel *fenc, pixel *p0, pixel *p1, pixel *p2, pixel *p3, int i_stride, int cost[4], int ly ) {

__m256i sum = _mm256_setzero_si256();

int i;

i_stride <<= 1;
for( i = 0; i < ly; i += 2 ) {

__m256i se = _mm256_load_si256( (__m256i *)(fenc) );
__m256i s0 = _mm256_loadu_si256( (__m256i *)(p0) );
__m256i s1 = _mm256_loadu_si256( (__m256i *)(p1) );
__m256i s2 = _mm256_loadu_si256( (__m256i *)(p2) );
__m256i s3 = _mm256_loadu_si256( (__m256i *)(p3) );

s0 = _mm256_sad_epu8( se, s0 );
s1 = _mm256_sad_epu8( se, s1 );
s2 = _mm256_sad_epu8( se, s2 );
s3 = _mm256_sad_epu8( se, s3 );

s0 = _mm256_hadd_epi32( s0, s1 );
s1 = _mm256_hadd_epi32( s2, s3 );
sum = _mm256_add_epi32( sum, _mm256_hadd_epi32( s0, s1 ) );

se = _mm256_load_si256( (__m256i *)(fenc + 32) );
s0 = _mm256_loadu_si256( (__m256i *)(p0 + 32) );
s1 = _mm256_loadu_si256( (__m256i *)(p1 + 32) );
s2 = _mm256_loadu_si256( (__m256i *)(p2 + 32) );
s3 = _mm256_loadu_si256( (__m256i *)(p3 + 32) );

s0 = _mm256_sad_epu8( se, s0 );
s1 = _mm256_sad_epu8( se, s1 );
s2 = _mm256_sad_epu8( se, s2 );
s3 = _mm256_sad_epu8( se, s3 );
}
We have cooperated with some leading media Service providers in China and optimized the video transcoding cluster’s performance. They have different transcoding requirements, and different types of the video sources, however, from the figure 6 to the figure 9, many time-consuming functions in those applications can be optimized by IA SIMD technology.

![Figure 7: Profiling Result of the Video Transcoding Application – Case II](image-url)
For example, Figure 9 is the profiling data on Strongene standard 1080p HEVC encoding scenario, 60% hot functions are running in SIMD SSE instructions, such as low-complexity motion compensation interpolation, transpose-free integer transform, butterfly Hadamard transform and the least-memory-redundancy SAD/SSD calculation.

AVX2 instruction could theoretically double the performance of previous 128b SSE code by 256b int computing, which is supported in Xeon Haswell generation platform, we can expect further great performance improvement when upgrade the SSE code to AVX2 on Haswell platform.

Based on above SIMD programming model and paradigms, customer can rewrite the hot functions in their video applications to pursue maximum performance increase.
3.1.2 SMT/HT Contribution for Media Processing

Simultaneous Multithreading (SMT), also called Hyper-threading (HT) technology is widely supported in the most of IA platforms, that makes the operating system addresses two virtual or logical cores for each physical core, and shares the resources between them when possible. The main function of hyper-threading is to decrease the number of dependent instructions on the pipeline. It offers performance benefits when CPU cores fully running in the heavy level, but not in every application such as that have the cores stay idle, in this case SMT technology will introduce the task/thread switching overhead.

Figure 10 demonstrates that multithreading can effectively improve the efficiency of the video transcoding applications for all kinds of the video formats and various bitrate requirements. And generally, 4-8 threads may approach the performance inflection point.

![Figure 10: Thread Efficiency in Video Transcoding Application](image)

We further conducted the multi tasks benchmarking to explore the best parallelism and simulate the real scenarios in the transcoding applications, as shown in the Table 2, generally the SMT/HT technology can contribute about 10-20% performance improvement in the multi tasks mode.

<table>
<thead>
<tr>
<th>Tasks</th>
<th>bit rate</th>
<th>SMT ON / SMT OFF (higher is better)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>250k</td>
<td>450k</td>
</tr>
<tr>
<td>1</td>
<td>0.922685</td>
<td>0.926111</td>
</tr>
<tr>
<td>4</td>
<td>1.017122</td>
<td>0.967597</td>
</tr>
<tr>
<td>8</td>
<td>1.119462</td>
<td>1.181345</td>
</tr>
<tr>
<td>12</td>
<td>1.204938</td>
<td>1.171228</td>
</tr>
<tr>
<td>16</td>
<td>1.1971</td>
<td>1.195339</td>
</tr>
</tbody>
</table>
Table 2: SMT/HT Contribution for Video Transcoding application

3.1.3 Other High Performance Tools and Library

To maximize IA platform’s capabilities and facilitate customer to utilize and deploy advanced IA technologies, Intel developed full set of high performance libraries, for all IA based client and server platforms, many OSs (Windows*, Linux*, OS X* and Android), and various of domains, such as system profiling, compiler, math kernel libs, cluster analysis, graphics SDK, and multithreading programming tools. We take the Intel® Compiler and IPP (Intel® Integrated Performance Primitives) for examples to demonstrate these high performance tools’ contribution for media cloud applications.

3.1.3.1 Intel® Compiler

ICC (Intel® Compiler)[5] generates the optimized code for all IA based platforms automatically, including the auto vectoring and paralleling, memory and cache line tuning, as well as serious of high level optimization, based on Intel architecture’s advanced features. It explores the most possible way to complete the task within the minimal CPU cycles, and compatible with Microsoft Visual C++ on Windows, GCC (GNU Compiler Collection) on Linux.

Figures 11 and 12 are two examples of ICC contribution to video transcoding applications, for H.264 and H.265 projects respectively. Generally, for that applications running on Intel architectures, ICC could help customers to have further performance improvement more easily and flexibly.
3.1.3.1 Intel® Integrated Performance Primitives

IPP (Intel® Integrated Performance Primitives) exploit the best thread-level parallelism and Intel architecture instruction set implementation for following applications and algorithms:

1) Image, video and audio processing
2) Data communication
3) Data compression and encryption
4) Signal processing, etc.
Here we take a common image/frame scaling workload to demonstrate how to use the ipp library to replace original implementation and achieve better performance on IA platform:

```c
#include "lanczos.h"
#include <ipp.h>
#include <time.h>
#include <stdlib.h>

//----------Original Scaling Implementation --------------------------

void original_scaling(IplImage *src, IplImage *dst, int width, int height)
{
    double x_factor;
    double y_factor;
    LanczosResizeFilter *resize_filter;
    int long span = 0;
    IplImage * filter_image = NULL;
    CvSize filter_size;
    unsigned int status;

    x_factor = (double)width/src->width;
    y_factor = (double)height/src->height;

    // set the value of filter structure //
    resize_filter = AcquireLanczosResizeFilter();
    filter_size.width = width;
    filter_size.height = height;

    //create temp matrix //
    if ((x_factor*y_factor) > WorkLoadFactor)
    {
        filter_size.width = width;
        filter_size.height = src->height;
        filter_image = cvCreateImage(filter_size, src->depth, src->nChannels);
    }
    else
    {
        filter_size.width = src->width;
        filter_size.height = height;
        filter_image = cvCreateImage(filter_size, src->depth, src->nChannels);
    }

    // compute pixxl of dest matrix//
    if ((x_factor*y_factor) > WorkLoadFactor)
    {
        span = (int long)filter_image->width + height;
        cv::Mat srcMat = cv::cvarrToMat(src);
        cv::Mat filterMat = cv::cvarrToMat(filter_image);
        cv::Mat dstMat = cv::cvarrToMat(dst);
        status = lanczosHorizontalFilter(resize_filter, &srcMat, &filterMat, x_factor, span);
        status &= lanczosVerticalFilter(resize_filter, &filterMat, &dstMat, y_factor, span);
    }
    else
    {
        span = (int long)filter_image->height + width;
        cv::Mat srcMat = cv::cvarrToMat(src);
        cv::Mat filterMat = cv::cvarrToMat(filter_image);
        cv::Mat dstMat = cv::cvarrToMat(dst);
        status = lanczosVerticalFilter(resize_filter, &srcMat, &filterMat, y_factor, span);
    }
}
```
status &= lanczosHorizontalFilter(resize_filter, &filterMat, &dstMat, x_factor, span);
}
// memory free Matrices
cvReleaseImage(&filter_image);
DestroyLanczosResizeFilter(resize_filter);

//IPP Code

void ipp_scaling(IplImage *src, IplImage *dst, int width, int height)
{
LanczosResizeFilter *resize_filter;
/* set the value of filter structure */
resize_filter = AcquireLanczosResizeFilter();
double x_factor;
double y_factor;
x_factor = (double)width/src->width;
y_factor = (double)height/src->height;

ippiSetNumThreads(1);

//define ipp parameters
IppiRect srcRoi = {0,0, src->width, src->height};
IppiRect dstRoi={0,0, width,height};
IppiSize srcSize = {src->width, src->height};
IppiSize dstSize = {width,height};

int interpolation = IPPI_INTER_LANCZOS;
int srcStep, dstStep;
int channel = src->nChannels;

//allocate memory
int BufferSize;
ippiResizeGetBufSize(srcRoi, dstRoi, channel, interpolation, &BufferSize);
Ipp8u* pBuffer=ippsMalloc_8u(BufferSize);

if(channel == 1) {
    //ippiConvert_32f8u_C1R((Ipp32f*)Temdst->imageData,
    Temdst->widthStep,(Ipp8u*)dst->imageData, dst->widthStep, dstSize, ippRndNear);
    ippiResizeSqrPixel_8u_C1R((Ipp8u*)src->imageData, srcSize, src->widthStep,
    srcRoi, (Ipp8u*)dst->imageData, dst->widthStep, dstRoi, x_factor, y_factor,0.0,
    0.0, interpolation, pBuffer);
}
else {
    //ippiConvert_32f8u_C3R((Ipp32f*)Temdst->imageData,
    Temdst->widthStep,(Ipp8u*)dst->imageData, dst->widthStep, dstSize, ippRndNear);
    ippiResizeSqrPixel_8u_C3R((Ipp8u*)src->imageData, srcSize, src->widthStep,
    srcRoi, (Ipp8u*)dst->imageData, dst->widthStep, dstRoi, x_factor, y_factor,0.0,0.0,
    0.0, interpolation, pBuffer);
}
ippsFree(pBuffer);
}
Figure 13 is the result of using ipp scaling api to optimize the original code. Since the IA SIMD/AVX instructions have been utilized and implemented well in IPP library, 1.67x performance speedup has been accomplished in this application.

Figure 13: IPP optimized libs for image/video processing

### 3.2 GPU based Hardware Transcoding Solution

Video transcoding involves converting one compressed video format to another. The process can also apply changes to the format and/or the video properties, like bit rate or resolution. Those processes include, reading the source video, decoding the video to the raw data, performing dedicated processing or enhancement (such as scaling, de-interlacing, editing and quality enhancement), encoding the raw data to specified format, and finally writing to the bit streaming or file.

As we illustrated in Section 2.2, in the real-time video transcoding applications, the GPU based HW video transcoding technology can boost the performance greatly in terms of the single task latency. Figure 14 is the general architecture and diagram of using the graphics to accelerate the video processing. In these applications, CPU is in charge of the audio and other auxiliary operations, the heavy load video functions will be transferred to the GPU part to take the advantage of the highly parallel capabilities on GPU, then all the data will be combined together to generate final output.
Figure 14: Video Processing with Graphics

Intel HD Graphics, also known as Intel GMA (Graphics Media Accelerator), integrated in the Intel® Core™ Processor and Xeon® Processor E3 platforms, has evolved to the fourth generation, featuring the excellent balance between video quality and the processing performance through a series of upgrade on both hardware architecture and drivers. In this section, we will introduce the Intel HD Graphics technology, describe the new high-performance hardware acceleration engine, and how to amplify the H.264/mpeg2 video transcoding and video effect processing.

3.2.1 Intel® Quick Sync Video Technology

Intel® Quick Sync Video[10] technology was first introduced with the 2nd generation Intel Core processor in 2011, aims to provide most efficient video transcoding solution by shared cache between CPU and GPU, fixed acceleration functions in GPU, and lower power technologies. Along with the generation to generation improvement, the 4th generation of Quick Sync Video technology in the Intel Xeon Processors and Intel Core processors platform has achieved great balance between video quality and processing performance/power through a serious of technology development like:

- More powerful EUs (execution units) based on the sliced architecture, each slice contains multi sub-slices of EUs and a faster media sampler with shared buffers/memory between them, and the high B/W access eDRAM channels.
- A scalable architecture with the flexibility to enable acceleration based on application demand, and programmable media-optimized EU (execution units)/samplers for high quality.
- A dedicated video quality engine, motion estimation engine, and entropy encoding accelerator to provide extensive video processing at low power consumption, such as the enhancements of De-Interlace and De-Noise.
- Fully accelerated decoder and bit packer for H.264, MPEG-2 & VC-1, and additional JPEG/MJPEG decode in the multi-format codec engine. These features
are on top of existing energy-efficient, high-performance AVC encode/decode that sustains multiple 4K and Ultra HD video streams.

The most time-consuming part in the video transcoding is the encoding functions. As demonstrated in the figure 15, Intel implements a high performance and flexible HW solution for the encoding:

- the fixed HW functions in the VDBOX (MFX: multi-format video codec) and VEBOX (video enhancement engine)
- programmable EUs and advanced samplers
- balance between performance, quality and power

“ENC” part operations are accelerated by the EU array and media samplers. “PAK” part operations are accelerated by the MFX engines in the media block. “ENC” and “PAK” execute concurrently to achieve the best performance.

Figure 15: Intel® Quick Sync Video Technology for the video transcoding

Intel Xeon processors with integrated GPU based hardware video transcoding solution can reach the excellent latency performance for single task scenario via HW fixed functions and high parallel EU capabilities. The 4th generation of Quick Sync Video technology has implemented a serious of scheme to improve video quality:

- hardware new features and algorithms: look ahead BRC, multi reference, multi predictors, trellis quantization, B pyramid, etc.
- fine tuned quality vs performance tradeoff control: 7 new distinctive target usages (TU) to control quality/performance. Developers can utilize the different level of TU to get the best tradeoff.
A lot of experiments have proved[11] that the new generation of Quick Sync Video technology achieves significant improvement at the HW video transcoding quality and performance. The TU based usage design optimizations provide significant quality performance headroom and enable a wider variety of applications, include quality sensitive video scenarios. Developers can take advantage of greater flexibility in the performance and quality to meet most of application requirement, from HD video conferencing to high quality video editing.

At the same time, more others are also on the continues improvement of the graphics architecture and software with the new video codec by Intel, such as full hardware acceleration of mpeg2 decoding and encoding, main profile and high profile of H.264 encoding, and upcoming H.265 codec implementation.

### 3.2.2 Graphics Driver and Software

Intel provides the full set of graphics driver and software to facilitate the developers to design and implement their own video applications:

- open source graphics driver[13]
- Intel® Media SDK[12]
- Intel® OpenCL SDK[14]

![Intel Software Stack for Graphics](image)

Figure 16: Intel Software Stack for Graphics

User can develop their own video application either from architecture drivers or higher SDK APIs. Video transcoding application has been enabled through Intel® Media SDK (MSDK):

- Software stack for PG’s video processing functions
Quality and performance modes provided for different usage cases. User can choose difference values for the option ‘u’ in MSDK, which specifies a trade-off between quality and speed: 1 = quality;4 = balance;7 = fast. Totally 7 levels for different scenarios. For further video quality requirement, user can enable the look ahead mode as well.

Flexible Encoder Infrastructure (FEI): Flexibility within the pipeline to allow differentiation and finer control of the encoding process

Asynchronous transcode flow for performance. Figure 17 demonstrates the performance improvement by asynchronous scheme for video transcoding application on both of fast and quality modes for three 480p, 720p and 1080p videos.

In Summary, Intel® HD Graphics and corresponding software provide the flexible video transcoding implementation on Processor Graphics (PG), the hardware fixed functions acceleration, and the scheme of balancing the quality and performance modes for different usage modules.

### 3.2.3 Case Study

A typical usage case is to encode the given image/frame to the target video format, like following diagram, GPU will take the YUV data to perform the dedicated video encoding task according to the target video format requirement and related bitrate and quality control.

![Figure 18: One Typical Video Encoding Processing via GPU](image-url)
Here is the example of how to implement it on Intel Graphics platform with both of open source driver and MSDK:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <ipp.h>
#include <ippcc.h>

……

//---------- general encoding process ------------------------
……

unsigned char* outbf;
outbf=(unsigned char *)malloc(width*height*2);
int pDstStep[3];
unsigned char *pDst[3];
……
//convert image to YUV format
pDst[0]=outbf;
pDst[1]=outbf+width*height;
ippiBGRToYCbCr420_709CSC_8u_AC4P3R(inb,width*4,pDst,pDstStep,roiSize);

int enc_size,offset;
struct timeval t_n,t_lstart;
unsigned char *enc_buf=(unsigned char *)malloc(width*height*2);
struct coded_buff cdbf;
cdbf.buff=enc_buf;

//encoding
encode_frame(outbf,&cdbf);

/*TSPack(&cdbf);*/
/*RTF sending or ts sending with ffmpeg */
/*rtp_sendpackage(&cdbf);*/

muxer_write_video_es(muxer, cdbf.buff, cdbf.length);

free(outbf);
free(enc_buf);
}
……

//--------- open source driver Code ------------------------
……

#include <va/va.h>
#include <va/va_enc_h264.h>
#include <va/va_x11.h>
#include "./vaenc.h"

……

int encode_frame(unsigned char *inputdata, struct coded_buff *avc_p) {

    int index;
    if (avcenc_context.current_input_surface == SID_INPUT_PICTURE_0)
        index = SID_INPUT_PICTURE_1;
    else
        index = SID_INPUT_PICTURE_0;
```
avcenc_context.upload_thread_param.yuv_p = inputdata;
avcenc_context.upload_thread_param.surface_id = surface_ids[index];

avcenc_context.upload_thread_value =
pthread_create(&avcenc_context.upload_thread_id,NULL,upload_thread_function,
(void*)&avcenc_context.upload_thread_param);

/*@  
static int const frame_type_pattern[][2] = { {SLICE_TYPE_I,1},
{SLICE_TYPE_P,3}, {SLICE_TYPE_P,3},{SLICE_TYPE_P,3},
{SLICE_TYPE_P,3}, {SLICE_TYPE_P,3},{SLICE_TYPE_P,3},
{SLICE_TYPE_P,3}, {SLICE_TYPE_P,3},{SLICE_TYPE_P,3},
{SLICE_TYPE_P,2} };
*/
if ( i_frame_only ) {
encode_picture(avc_p,enc_frame_number, frame_number, frame_number==0,
SLICE_TYPE_I, 0, frame_number+1);
frame_number++;  
enc_frame_number++;  
} else if ( i_p_frame_only ) {
if ( (frame_number % intra_period) == 0 ) {
encode_picture(avc_p,enc_frame_number, frame_number, frame_number==0,
SLICE_TYPE_I, 0, frame_number+1);
frame_number++;  
enc_frame_number++; 
} else {
encode_picture(avc_p,enc_frame_number, frame_number, frame_number==0,
SLICE_TYPE_P, 0, frame_number+1);
frame_number++;  
enc_frame_number++; 
}
}
else { // follow the i,p,b pattern
static int fcurrent = 0;
int fnext;

fcurrent = fcurrent % (sizeof(frame_type_pattern)/sizeof(int[2]));
fnext = (fcurrent+1) % (sizeof(frame_type_pattern)/sizeof(int[2]));

if ( frame_type_pattern[fcurrent][0] == SLICE_TYPE_I ) {
encode_picture(yuv_fp, avc_fp,enc_frame_number, f, f==0, SLICE_TYPE_I, 0,
frame_type_pattern[fcurrent][0], f+frame_type_pattern[fnext][0], f++,
enc_frame_number++;
} else {
encode_pb_pictures(yuv_fp, avc_fp, f,
frame_type_pattern[fcurrent][1]-1,
frame_type_pattern[fcurrent][1], f += frame_type_pattern[fnext][1]-1);
enc_frame_number++;
}

fcurrent++; 
}
return 0;
}

int close_encoder()  
{
    release_encode_resource();
}
4. Industry Deployment and Direction

IA platforms have taken the leadership in the media cloud domain via architecture design, rigorous manufacture procedure and performance and service. We help many media ISVs to evaluate the media processing clusters’ capabilities, tuning the performance, and figure
out more efficient solutions. As the new technologies emerging constantly in this industry, we also keep investigating the following directions.

### 4.1 Distributed Processing Platform

Most of the media processing clusters are distributed platforms, how to allocate the distributed resource to complete the processing task in the more effective way is a big challenge for the media ISVs. As we illustrated in Figure 10 in Section 3.1.2, traditional video transcoding application can utilize 4-8 cores’ computing resource in multi-thread mode, however, with upgrades of CPU architecture and hardware, Intel® Xeon® processor platform (Haswell) can provide up to 18 physical cores per CPU, 36 logical cores when Intel® HT Technology is enabled. Therefore an intelligent distributed media processing solution is very critical to fully utilize capabilities all processor cores, and have all computing nodes working in parallel.

Slicing the media file into several smaller pieces and launching multi parallel task could be a good direction. Media files are split to several pieces, that could also help saving the network BW for buffering and playing video in parallel, then a flexible job scheduler scheme can control and monitor all nodes’ working status, coordinate the distributed tasks, and balance the resources. Lots of customers have been developing their own distributed processing platform recently, we are also working with some of them to design a more intelligent distributed solution with the adaptive analysis on both of media content complexity and computing nodes capabilities based on the IA platform.

### 4.2 HEVC/H.265 Codec

Video coding standards have been evolved primarily through the development of the well-known ITU-T and ISO/IEC standards. The ITU-T produced H.261 and H.263, ISO/IEC produced MPEG-1 and MPEG-4 Visual, and the two organizations jointly produced the H.262/MPEG-2 Video and H.264/MPEG-4 Advanced Video Coding (AVC) standards [1]. H.265/HEVC (High-Efficiency Video Coding), introduced in 2013, is the latest video codec standard developed by ISO / IEC and ITU-T, aimed to maximize compression capability and reduce data loss.

HEVC/H.265 technology, the next video codec revolution, helps the media service providers providing high-quality video with less bandwidth, and supporting 4k (4096×2160) and 8k (7680×4320) SHD video service. H.265/HEVC doubles the compression ratio compared to the previous H.264/AVC standard, but in the same subjective quality, as shown in the Table 3, that will save lots of the cost but provide the better experience to end users.

<table>
<thead>
<tr>
<th>File: BQTerrance_1920x1080_60.yuv</th>
<th>Codec</th>
<th>Size(byte)</th>
<th>Bitrate(kbps)</th>
<th>PSNR_Y/U/V(db)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution: 1920x1080 Size: 1869Mbyte. 622080 kbps</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Platform: E5-2697 v2 @2.70GHz, RAM 64GB DDR3-1867, QPI 8.0 GT/s
OS/SW: Red Hat 6.4, kernel 2.6.32, gcc v4.4.7, ffmpeg v2.0.1, Lentoid HEVC Encoder r2096 linux

<table>
<thead>
<tr>
<th>Codec</th>
<th>Throughput (frames/sec)</th>
<th>FPS (1080p)</th>
<th>FPS (1920p)</th>
<th>FPS (2560p)</th>
</tr>
</thead>
<tbody>
<tr>
<td>H.264</td>
<td>12254696</td>
<td>4078.1</td>
<td>32.311/39.369/42.043</td>
<td></td>
</tr>
<tr>
<td>H.265</td>
<td>6215615</td>
<td>2064.28</td>
<td>34.016/39.822/42.141</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: H.264 and H.265 codec performance compare

However, the computing complexity of H.265/HEVC codec is far more 4 times than previous H.264/MPEG, it raises unprecedented processing requirements to the backend server platform, which attract world-wide multi groups/agencies to optimize for better performance, push to real deployment. Several open sourced projects are:

a) OpenHEVC(currently HM10.0 compatible, and did some optimization on decoder)
   https://github.com/OpenHEVC/openHEVC

b) x265(compatible with HM, and did optimization on parallel & SIMD)
   http://code.google.com/p/x265/
   https://bitbucket.org/multicoreware/x265/wiki/Home

We worked with a leading codec provider to optimize one H.265 codec before[9], accomplished a CPU based real-time software HEVC encoding solution on Intel® server platform with our new platform technologies. As GPU based hardware acceleration solution also emerging and becoming mature, H.265 could be the major standard in the coming decade.

4.3 Data Mining and Others

Media files contain lots of information indeed, like the time and location, the people and their behavior, and even the clothes they wear. Data mining and machine learning can be conducted based on those kind of information, extract the useful information, make some dedicated analyze and draw the necessary conclusion, such as help finding the suspects or losing people, analyze the relationship between the videos, and promote more business models based on people’s favorite. In addition, as Internet of Things (IOT) business and application are maturing, lots of media data processing also will be involved in the IOT system.

For the high dimensional and high parallel media processing, extracting the dedicated content and analyzing the media data will consume lots of computing resource. Designing the high performance algorithm and platform will be trend and focus of the industry over the next few years.

5. Performance Tuning Methodology in Media Cloud Domain
To summarize all the performance tuning work in the previous sections, we have the
serials of IA tuning methodology in media cloud domain. As illustrated in Table 4, we
take the following tuning steps to optimize the media cloud applications orderly, from
system level to micro-arch level.

<table>
<thead>
<tr>
<th>Order</th>
<th>Optimize level</th>
<th>Optimize Target</th>
<th>Key areas to investigate</th>
</tr>
</thead>
</table>
| 1     | High: System Level Tuning    | By figuring out system level bottleneck to improve the performance             | • Network problem
• Disk issue
• Memory issue
• OS related issue
• System load balance and NUMA
• GPU bound I/O issue
• GPU bound memory traffic balance
• CPU & GPU load balance |
| 2     | Medium: Application Level Tuning | By improving the implementation of applications’ algorithm and the library to boost the performance | • Thread implementation
• Process/Thread locks
• Choose optimized media libs or APIs
• Data structure tuning
• Core media algorithms tuning
• Graphics libs tuning |
| 3     | Low: Micro-arch level tuning | By improving how the application running in specific platform from micro-code level | • Data/operation localization (Cache efficiency)
• Data alignment
• Hot code vectorization /SIMD
• micro-arch tuning on both of CPU and GPU parts |

Table 4: IA Platform Performance Tuning Methodology in media Cloud

## 5.1 System Level Tuning

The first step is to analyze whether the usage model has any system level bottleneck, like
the I/O disk issue, network bottleneck, and any memory problem. We know that media
cloud applications are computing intensive workload, that means all CPU cores should run heavily, if you find any cores idle, you need to check the disk I/O, network and memory issues firstly, to figure out whether the system has any components blocked the normal computing process. Following are some examples of using the tools to monitor the system status, such as the Linux commands top, sar, iostat, vmstat, and etc. For GPU part, gpu_top and Intel® Graphics Performance Analyzers (GPA) [15] can help profiling the graphics performance.

```text
avg-cpu: %user  %nice  %system %iowait  %steal  %idle
         0.00    0.00    0.01    0.00    0.00   99.99
```
Taking memory bandwidth for example, the media data, especially for the video, consume lots of memory bandwidth during computing, one more memory channel will boost the performance distinctly for both of CPU SW solution and GPU HW solution.
As shown in Figures 19 and 20, adding one more memory channel can contribute around 40% performance speedup in these two video transcoding scenarios. Therefore keeping the memory channels balanced and fully loaded could be very important for media cloud applications.
5.2 Application Level Tuning

After finishing system level performance tuning, we need to investigate the application and algorithms further. As we have illustrated in the previous Sections 3.1 and 3.2, IA build series of high performance libraries and tools to facilitate customers maximizing the IA platform performance more easily and flexibly.

The most important thing during the application tuning is conducting system profiling properly, which will help to find the bottleneck and hot function correctly. Intel® VTune™ Amplifier is a very accurate tool to analyze the IA platform applications, which will generate a rich set of performance report into hotspots, threading, locks & waits, bandwidth and more system units. Customer can use the friendly UI and powerful analysis tool to sort, filter and visualize results on the timeline, and trace the source code conveniently. Figure 6 to figure 9 in the section 3.1 are the examples we used vtune during the performance tuning work. From the performance perspective, only the work done at the hotspots are effective.

As demonstrated in Figure 21, Intel provides full set of high performance libraries on both of CPU and GPU, for all IA based platforms and various of domains. When we do the system level performance tuning at IA platforms, we could check the high performance libraries, to see whether the system have the optimized implementation there:

- Intel® Math Kernel Library: deliver high performance at math libraries, such as matrix multiplication, FFT calculation, and so on.
- IPP(Intel® Integrated Performance Primitives): exploit very effective implementation from thread-level parallelism and IA architecture instruction set for the image and video processing, data communication and compression, and signal processing, etc.
- Intel® Media SDK: deliver very effective implementation for IA Graphics on both of windows and linux platforms (some versions of kernel).
- Intel® OpenCL SDK: provide the preferable development environment for OpenCL applications at Intel Graphics platforms.
- TBB (Intel® Threading Building Blocks) provide better task parallelism mechanism for customer developing the multi-task and multi thread applications on IA multi-core platforms. Its memory management and threading locking engine have achieved pretty good performance for lots of customer applications.
Beyond the IA specific application tuning methodology above and the works we demonstrated in section 3.1 and 3.2, we also need to figure out the challenges by the media processing and algorithm itself, to explore the possibilities of improving the performance. For example, Table 5 illustrates the performance sensitive properties in traditional video transcoding applications, we will do tuning these properties according to the applications requirement and achieved performance improvement shown in the figure 22.

<table>
<thead>
<tr>
<th>Transcoding Properties</th>
<th>Properties Description and Tuning Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>-me_method</td>
<td>Motion Estimation method used in encoding (zero : zero motion estimation (fastest) full : full motion estimation (slowest) epzs : EPZS motion estimation (default) esa /esa /dia /log /phods /x1 /hex /umh )</td>
</tr>
<tr>
<td>-me_range:</td>
<td>max range of the motion search (4-16)</td>
</tr>
<tr>
<td>-subq:</td>
<td>the algorithms are used for both sub-pixel motion searching and partition decision (1-9; 1: Fastest, but extremely low quality)</td>
</tr>
<tr>
<td>-b_qfactor:</td>
<td>&lt;float&gt; Qscale difference between P-frames and B-frames.</td>
</tr>
<tr>
<td>-bf</td>
<td>&lt;int&gt; number of bframes, 3is best in terms of the performance</td>
</tr>
<tr>
<td>-g</td>
<td>&lt;frames&gt; &lt;keyint&gt; Keyframe interval, GOP length. This determines the maximum distance between I-frames. Very high GOP lengths will result in slightly more efficient compression, but will make seeking in the video somewhat more difficult.</td>
</tr>
</tbody>
</table>

Table 5: Performance Sensitive Properties at Traditional Video Transcoding Application
5.3 Micro-Arch Level Tuning

Next step, we need to focus on the micro-Arch tuning work, which needs full understanding of the platform structure and CPU architecture.

In some cases, the most time-consuming hot codes can’t be optimized by the compiler automatically, and there’s also no similar high performance API which can be adopted directly, we need to analyze the code and rewrite them with SIMD (Single Instruction Multi Data) instructions manually according to the programmer guide and related documents. If the scalar code can be rewritten with SIMD fully, even with the assembly tuning, a significant performance boost can be expected.

Since most of the time-consuming video and image functions located to the block based data intensive computing, which can be optimized by the x86 platform SIMD vectorization instructions. SIMD instructions process multi set data within one single CPU cycle, which will greatly improve the data throughput and execution efficiency. As we illustrated in the figure 23, “n” sets of data can be computed in one step by using the vector register, therefore we can boost the original scalar computing by “n” times theoretically.
Figure 23: SIMD methodology

Here the value of n depends on the vector width of the system and the data type in the application:

\[ n = \frac{\text{vector width}}{\text{length of data type}} \]

SIMD has been widely supported at x86 processors, evolving from MMX, SSE, AVX, to AVX2 at different x86 platform generations.

<table>
<thead>
<tr>
<th>SIMD</th>
<th>MMX</th>
<th>SSE</th>
<th>SSE2</th>
<th>SSE4.2</th>
<th>AVX</th>
<th>AVX2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support Platform</td>
<td>Pentium</td>
<td>Pentium III</td>
<td>Pentium 4</td>
<td>Nehalem</td>
<td>SandyBridge</td>
<td>Haswell</td>
</tr>
<tr>
<td>Vector Width</td>
<td>64b</td>
<td>128b</td>
<td>128b</td>
<td>128b</td>
<td>256b</td>
<td>256b</td>
</tr>
</tbody>
</table>

Table 6: SIMD Instruction Evolution

- MMX: 8x 64-bit registers (MM0 … MM7)
- SSE: 8x 128-bit registers (XMM0 … XMM7), 4x Single Precision FP per XMM register
- SSE2: wider integer vector width, 128b; double precision FP per XMM register
- SSE3: Incremental instructions over SSE2
• SSE4 (4.1 and 4.2): 16x 128 bit register (XMM0-XMM15), and new instructions (47 + 7 new instructions)
• AVX: 256 bit wide SIMD for float computing, and 16x256 bit wide (ymm0-ymm16)
• AVX2: 256 bit SIMD for integer computing, and new instructions for FMA(Fused Multiply Add), gather instructions, and extended instructions

As we have demonstrated in the section 3.1, the micro-arch level performance tuning, especially for the SIMD vectorization can boost the performance significantly, and have been widely adopted in the media cloud domain.

6. Summary

Media cloud applications are getting more and more popular and being the important part in the traditional datacenter as well as the mobile internet industry. For these data and computing intensive workloads, IA platforms build the hardware and software ecosystem on both of CPU and GPU platforms, for the latency, throughput, and quality sensitive scenarios. As the new business and usage modules emerging constantly, and IA platform upgrading stably, more and more media applications will get benefit from high performance and high reliability of IA technologies definitely.

Reference

[2] High Efficiency Video Coding (HEVC) text specification draft 10, JCTVC-L1003_v34