Using Intel® VTune™ Performance Analyzer and Intel® Integrated Performance Primitives for Real-time Media Optimization

Thu, Feb 9, 2012

By Sam Siewert

Real-time Media and Encoding Tools

Real-time media has become pervasive with Web 2.0 viral video services, Internet Protocol Television (IPTV), mobile media for Web-enabled phones, and interactive systems for digital cable. Digital content creation, production, delivery, and consumption-increasingly in high definition (HD) relative to standard definition (SD) television-is changing communication, with more on-demand services and high demand for digital media in a wide range of resolutions, encoding formats, and transport formats.

This article provides a brief overview of digital media concepts and software tools used for encoding video and audio in compressed formats for playback and transport. Using an open source encoder with a simple workflow to trans-code MPEG4 content into MPEG2, with debugging symbols included, moving on to cover the use of the Intel® VTune™ profiling tools to understand where processor utilization hot-spots are in the software in terms of number of instructions and efficiency in the execution of those instructions. Tracing down a specific hot-spot, I examine cache misses that contribute to inefficiency. Finally, examining the specific hot-spot, I find that it makes use of the *Intel® Streaming SIMD (Single Instruction Multiple Data) Extensions*, indicating that the code has employed the extensions to speed-up this section of code already, but also indicating a data cache inefficiency. The article reviews methods to improve cache efficiency, as well as both periodic and event-based profiling methods. In general, the paper provides an introduction to using Intel® VTune™ tools with digital media encoding software and demonstrates the importance of using *Intel® Streaming SIMD Extensions* (Intel® SSE) and understanding where code hot-spots are and methods that can be used to make execution of code in hot-spots most efficient.

This article focuses on video, because it has an inherently high data rate based upon the number of bits per pixel (picture element), number of pixels per frame (picture), and frames per second for full-motion video. The typical uncompressed SD encoding of National Television Standards Council (NTSC) video-what everyone knows as *TV*-has 29.97 interlaced frames per second typically digitized into 720×480 24-bit color-encoded pixels (8 bits for red, green, and blue [RGB]) at the same frame rate, which is right around a 30 megabytes (MB)/sec data rate if it was not further encoded with compression into a format like MPEG2. HD 1080×1920 resolution at 29.97 frames per second (fps) with the same 24-bit pixel encoding is about 180 MB/sec. Without encoders and decoders, video transport for streaming or even download would clearly not be practical, even with today's broadband networks.

Encoding and decoding is obviously necessary for video content delivery and consumption with players on both desktop and mobile computing devices. I first review a useful open source code base and tool called ffmpeg, which can encode a wide variety of original content into many compressed formats, including MPEG, from uncompressed Portable Pixmap (PPM) frame sequence formats as well as transcode from one encoding to another. Working through a few simple exercises to encode and transcode some open source content, you'll see that this is a very CPU-intensive task. To better understand where that CPU time is being spent and why encoding and transcoding is so CPU intensive, you'll use the Intel® VTune™ Performance

Analyzer to profile the ffmpeg code that you build for Windows*. Finally, I take a quick look at the use of Intel® Integrated Performance Primitives (Intel® IPP) and how you might optimize real-time media encoding tools like `ffmpeg`.

MPEG Encoding and Encoding Tools

Historically, digital video content came from studios; but today, most anyone can dabble in content creation and production using HD and SD cameras, a home computer, and some basic post-production tools to edit digital video and audio streams, and then encode them into deliverable playback formats or streaming formats. For this paper, please use your own content or download some open source content from one of these open content sources:

- http://www.bigbuckbunny.org/index.php/download/ (http://www.bigbuckbunny.org/index.php/download/) (MPEG4 PSs)
- http://orange.blender.org/download (http://orange.blender.org/download) (MPEG2 and MPEG4 PSs)
- http://www.w6rz.net/ (http://www.w6rz.net/) (a wide variety of PSs and TSs as well as test data)

Hopefully, you'll find this project not only informative but also entertaining.

A Brief Overview of Encoding Program and Transport Streams

PSs are container formats like Audio Video Interleave (AVI) and MPEG PSs. They encapsulate a number of elementary streams (ESs) composed of only video or audio into one compressed format that can include multiple ESs. Often, a video and an audio ES are multiplexed together to create what we all think of as a *movie*. These container formats or PSs work well for file download and playback with a media player off of local storage devices like disk, DVD, or flash. They typically include time stamps to guide the player on how fast to decode (Decode Time Stamps, or DTSs) and how fast to present (Presentation Time Stamps, or PTSs) to an output video monitor.

TSs are used to encapsulate multiple PSs into a format designed for network delivery-188-byte packets for MPEG. A TS includes the ability to provide multiple video and audio programs, Program Specific Information (PSI) or guide data, and additional timing used to keep decoders synchronized with real-time streaming services (the Program Clock Reference, or PCR). So, TSs often included multiple programs that are "packetized" ESs (PES).

Either way, the audio and video must first be encoded into ESs that can be packetized for further encapsulation into a PS or TS for delivery or playback. The encoding involves steps to compress video that include:

- Pixel compression, often for 8 bits for each color band RGB for 24-bit pixels down to 16-bits or less using color space sub-sampling so that luminance is provided more frequently than chrominance data-for example, YUV 4:2:2, where the *Y* (luminance) is sampled in every pixel and the *UV* (color RGB difference signals) are sub-sampled so that in each 4-pixel subset of an image, two UV color difference samples (Cr and Cb) are included every other pixel, also known as

YCrCb format. Computationally, this is a fairly simple in-memory reformat of data.

- Frame, 8×8 pixel macroblock encoding, where a Discrete Cosine Transform (DCT) is first applied to each block, followed by Quantization, and finally Huffman Encoding using the Zig-Zag pattern. The details behind this process go beyond the scope of this paper, but the key is to understand the complexity and basic sequence of mathematical and algorithmic operators applied to each video frame.
- Sequence, where temporal redundancy is removed in a Group of Pictures (GOP) so that an I-frame (compressed) is at the beginning of each GOP, but change-only data is encoded into P frames, with B frames between I and P that include only changes between surrounding I and P frames. Here again, this is a somewhat complex reformatting of data in a larger sequence in memory.

So, encoding is fairly memory and CPU intensive and must access large files. For single-stream encoding from SD, uncompressed video data, or from another compressed HD format, the bottleneck is most often the CPU. For encoding from uncompressed HD formats like 1080×1920, the bottleneck may also be the storage subsystem. Here, I focus on single-stream encoding and the CPU bottleneck. You would expect that a large portion of the CPU time would be spent in functions related to encoding and decoding, which you'll see in the Intel® VTune™ analyzer profile.

Open Source Encoding Tools for Linux* or Windows*
A huge variety of open source and for-purchase post-production encoding tools exist for both Linux* and Windows*. This paper focuses on encoding and transcoding (going from one encoded format like MPEG2 to another, like MPEG4). You'll use the Big Buck Bunny* 720×1280 24-fps open source content, which you can download to understand digital media encoding, how it uses CPU resources, and how you might optimize it.

You can download the ffmpeg tool used for transcoding in this paper from http://ffmpeg.arrozcru.org/builds/ (http://ffmpeg.arrozcru.org/builds/). I built Subversion revision 16537 of ffmpeg for this article using the MinGW tools and MSYS bash shell on my Windows laptop. The build went off without problems using the recommended build procedure in my MSYS shell:

```
./configure --enable-debug --enable-memalign-hack --extra-cflags="-fno-co
make
make install
```

The `--enable-debug` element is important for Intel® VTune™ Performance Analyzer so that you can drill down to the code hotspots, with symbols to identify functions where the most time is spent.

A Basic Encoding Workflow

An *encoding workflow* simply defines an input source file, the encoding operations to apply to it, and an output result file. The `ffmpeg` tool has many options for both audio and video encoding from either uncompressed original sources or already-encoded sources to many different encodings and file formats. Issuing `ffmpeg -formats` on the command line lists all formats available. For the basic resource usage and CPU profiling example in this paper, I transcoded the *Big Buck Bunny** AVI format file, first removing all audio with the command:

```
ffmpeg -i big_buck_bunny_720p_surround.avi -an -vcodec copy big_buck_no
```

Then, to study resource usage during transcoding, I used the following command to transcode a specific number of seconds of the MPEG4 video:

```
ffmpeg -t 10 -i big_buck_no_audio.mp4 -vcodec mpeg2video big_buck_no_audi
```

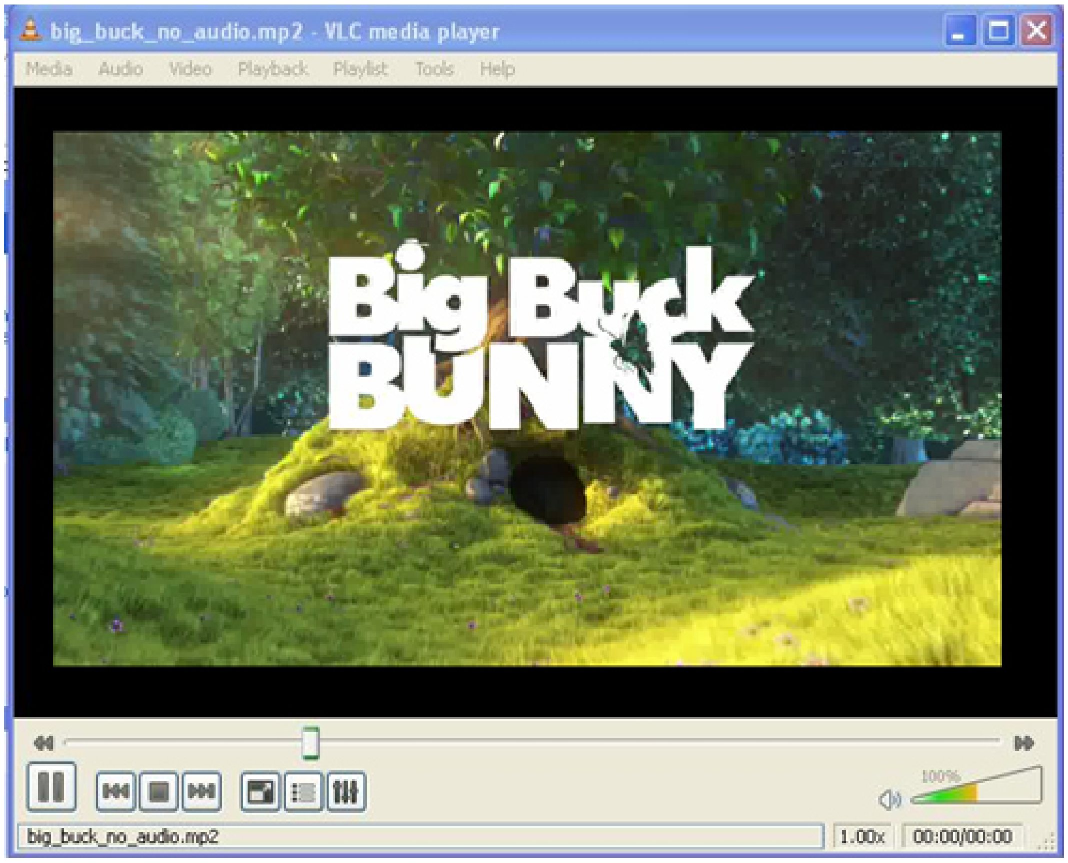
The `-t` 10 argument specifies 10 seconds transcoding the MPEG4 input into an MPEG2 output. The output for the 10-second transcode run looks like this:

```
Sam Siewert@SAM-LAPTOP /c/content/open-source
$ ffmpeg -t 10 -i big_buck_no_audio.mp4 -vcodec mpeg2video big_buck_no_au
FFmpeg version UNKNOWN, Copyright (c) 2000-2009 Fabrice Bellard, et al.
  configuration: --enable-memalign-hack --extra-cflags=-fno-common
  libavutil     49.12. 0 / 49.12. 0
  libavcodec    52.10. 0 / 52.10. 0
  libavformat   52.23. 1 / 52.23. 1
  libavdevice   52. 1. 0 / 52. 1. 0
  built on May  4 2009 02:12:23, gcc: 3.4.5 (mingw-vista special r3)
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'big_buck_no_audio.mp4':
  Duration: 00:09:56.45, start: 0.000000, bitrate: 4003 kb/s
    Stream #0.0(und): Video: mpeg4, yuv420p, 1280x720 [PAR 1:1 DAR 16:9],
Output #0, mp2, to 'big_buck_no_audio_10s.mp2':
    Stream #0.0(und): Video: mpeg2video, yuv420p, 1280x720 [PAR 1:1 DAR 1
Stream mapping:
  Stream #0.0 -> #0.0
Press [q] to stop encoding
frame=  240 fps= 40 q=31.0 Lsize=    1442kB time=9.96 bitrate=1185.9kbits
video:1442kB audio:0kB global headers:0kB muxing overhead 0.000000%
```

Previewing the resulting encoded output files is always an important step, because ultimately, this content will be delivered for playback, and eyeballs are still the best judges of video quality. An excellent open source option for preview and playback is Video LAN Client* (VLC), which has the capability to play back PS or AVI files as well as stream or play digital media TSs or PSs over User Datagram Protocol (UDP) or Real-Time Transport Protocol (RTP). You can download the VLC server/client for Windows or Linux as binaries or source to build from [http://www.videolan.org/vlc/ (http://www.videolan.org/vlc/)](http://www.videolan.org/vlc/).

Figure 1 shows an example VLC preview window with a scene from the *Big Buck Bunny* content that I transcoded to MPEG2 from MPEG4 and removed audio from with the following commands:

```
ffmpeg -i big_buck_bunny_720p_surround.avi -an -vcodec copy big_buck_no_a
ffmpeg -i big_buck_no_audio.mp4 -vcodec mpeg2 big_buck_no_audio.mp2
```


big_buck_no_audio.mp2 - VLC media player
Media Audio Video Playback Playlist Tools Help
Big Buck BUNNY
100%
big_buck_no_audio.mp2
1.00x
00:00/00:00

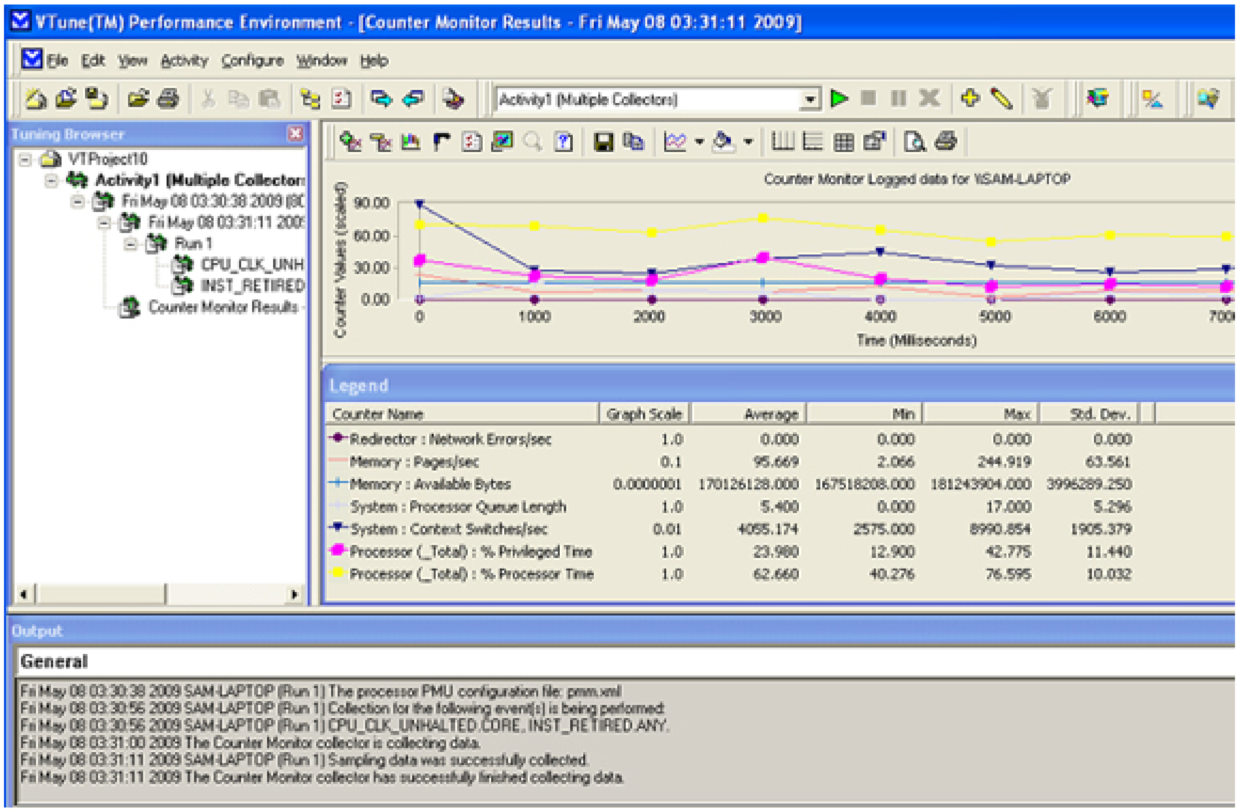
(c) copyright Blender Foundation | www.bigbuckbunny.org

Figure 1. Big Buck Bunny preview with VLC

Using Intel® VTune™ Performance Analyzer to Profile Encoding

You can download Intel® VTune™ Performance Analyzer for trial from Intel at /en-us/ (/en-us/). The trial is good for 30 days and will allow you to find hotspots in your code or in open source code like ffmpeg so that you can understand which functions and even which specific lines of code take the most time during execution. Intel® VTune™ Performance Analyzer works by using the x86 Performance Monitoring Unit (PMU), a hardware performance profiling unit built into x86 cores. You can program the PMU to raise an interrupt on a periodic basis to sample the Instruction Pointer address and key counters such as cycle count, instruction count, and number of cache misses since the last interrupt. Intel® VTune™ Performance Analyzer is ideally suited to software such as media encoders, because the work includes repetition of common functions used to encode or decode each frame and frames in each GOP and employs common functions and primitives repeatedly. Because of repetition in the workload, Intel® VTune™ analyzer periodic sampling profiles the code accurately as long as a test of significant duration is run. The basic concept is that if you randomly sampled what you do during the week enough times, you would correctly determine the amount of time you commute to work, sit at your desk, get up to get coffee, commute home, and so on, which might help point out where you're inefficient in your work day and find hotspots where you waste time so that perhaps you could work fewer hours, get more done, and head home earlier, even avoiding commute traffic. This is the basic idea of Intel® VTune™ Performance Tools analysis.

You run Intel® VTune™ tools on the ffmpeg_g.exe file, which has symbols so that you can drill down into the profile with function names and with path information to the full C source code you built with in the MSYS MinGW environment in Windows. Figure 2 shows the resulting analysis by Intel® VTune™ Performance Environment for basic counter monitors, including CPU time.

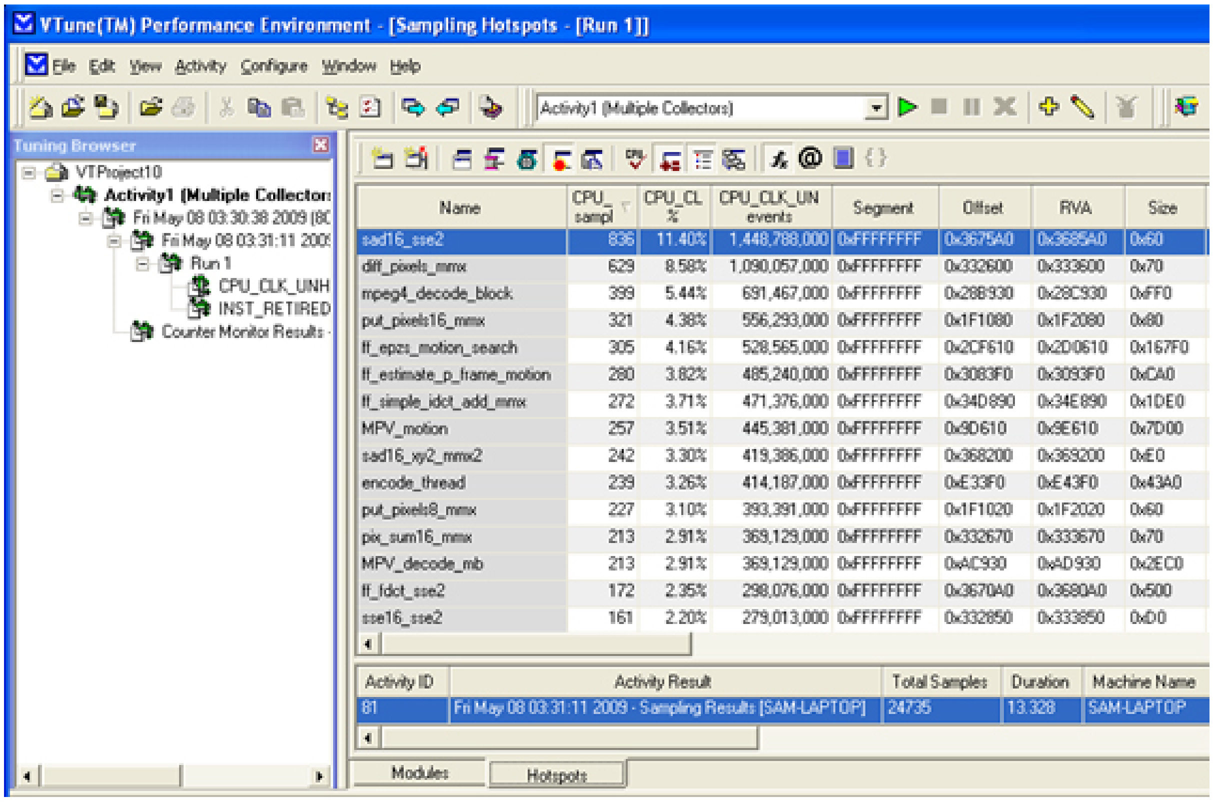

VTune(TM) Performance Environment - [Counter Monitor Results - Fri May 08 03:31:11 2009]
File Edit View Activity Configure Window Help
Activity1 (Multiple Collectors)
Tuning Browser
VTProject10
Activity1 (Multiple Collectors
Fri May 08 03:30:38 2009
Fri May 08 03:31:11 2009
Run 1
CPU_CLK_UNH
INST_RETIRED
Counter Monitor Results
Counter Monitor Logged data for \\SAM-LAPTOP
Counter Values (scaled)
90.00
60.00
30.00
0.00
0
1000
2000
3000
4000
5000
6000
7000
Time (Milliseconds)
Legend
Output
General

Counter Name	Graph Scale	Average	Min	Max	Std. Dev.
Redirector : Network Errors/sec	1.0	0.000	0.000	0.000	0.000
Memory : Pages/sec	0.1	95.669	2.066	244.919	63.561
Memory : Available Bytes	0.0000001	170126128.000	167518208.000	181243904.000	3996289.250
System : Processor Queue Length	1.0	5.400	0.000	17.000	5.296
System : Context Switches/sec	0.01	4055.174	2575.000	8990.854	1905.379
Processor (_Total) : % Privileged Time	1.0	23.980	12.900	42.775	11.440
Processor (_Total) : % Processor Time	1.0	62.660	40.276	76.595	10.032

```
Fri May 08 03:30:38 2009 SAM-LAPTOP (Run 1) The processor PMU configuration file: pmm.xml
Fri May 08 03:30:56 2009 SAM-LAPTOP (Run 1) Collection for the following event(s) is being performed:
Fri May 08 03:30:56 2009 SAM-LAPTOP (Run 1) CPU_CLK_UNHALTED.CORE, INST_RETIRED.ANY.
Fri May 08 03:31:00 2009 The Counter Monitor collector is collecting data.
Fri May 08 03:31:11 2009 SAM-LAPTOP (Run 1) Sampling data was successfully collected.
Fri May 08 03:31:11 2009 The Counter Monitor collector has successfully finished collecting data.
```

Figure 2. Counter monitor results

The data that is more useful than the basic counter monitors is the drill-down by program, thread, module, and finally source code function in terms of where most of the time is spent. Figure 3 shows the top functions where most of the time is spent, which matches the expectation that most of it would be in the encoding and decoding functions and related primitives.

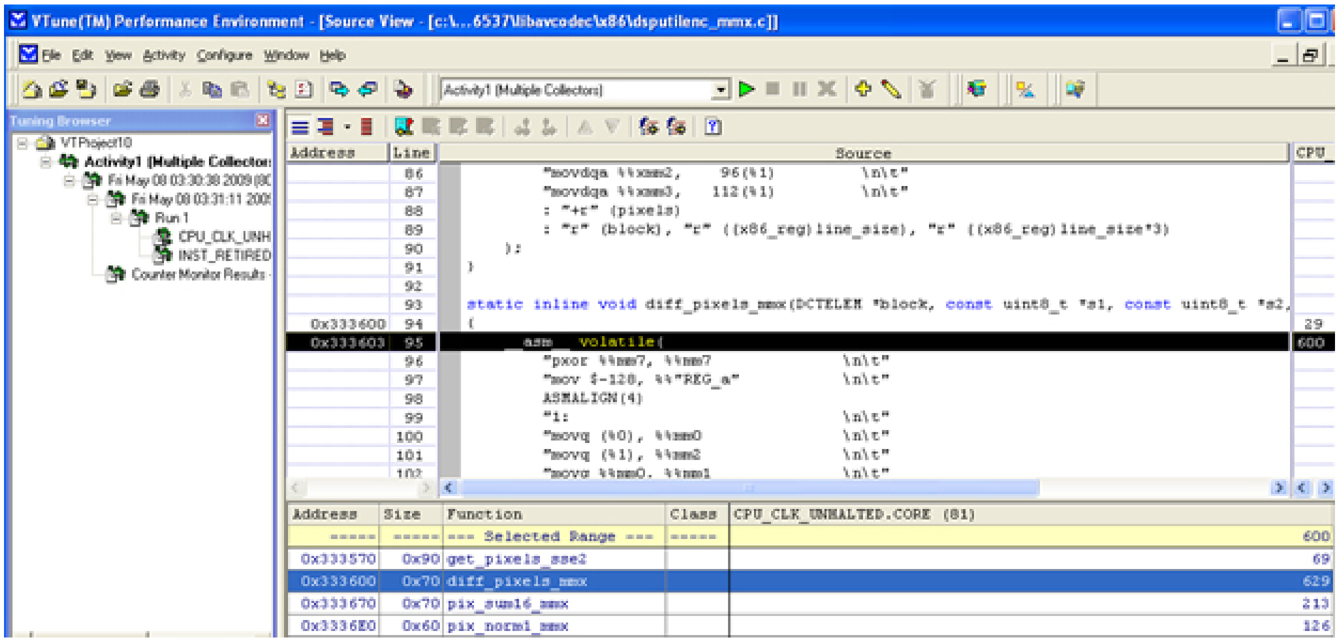


Name	CPU_ sampl	CPU_CL %	CPU_CLK_UN events	Segment	Offset	RVA	Size
sad16_sse2	836	11.40%	1,448,788,000	0xFFFFFFFF	0x3675A0	0x3685A0	0x60
diff_pixels_mmx	629	8.58%	1,090,057,000	0xFFFFFFFF	0x332600	0x333600	0x70
mpeg4_decode_block	399	5.44%	691,467,000	0xFFFFFFFF	0x28B930	0x28C930	0xFF0
put_pixels16_mmx	321	4.38%	556,293,000	0xFFFFFFFF	0x1F1080	0x1F2080	0x80
ff_epzs_motion_search	305	4.16%	528,565,000	0xFFFFFFFF	0x2CF610	0x2D0610	0x167F0
ff_estimate_p_frame_motion	280	3.82%	485,240,000	0xFFFFFFFF	0x3083F0	0x3093F0	0xCA0
ff_simple_idct_add_mmx	272	3.71%	471,376,000	0xFFFFFFFF	0x34D890	0x34E890	0x1DE0
MPV_motion	257	3.51%	445,381,000	0xFFFFFFFF	0x9D610	0x9E610	0x7D00
sad16_xy2_mmx2	242	3.30%	419,386,000	0xFFFFFFFF	0x368200	0x369200	0xE0
encode_thread	239	3.26%	414,187,000	0xFFFFFFFF	0xE33F0	0xE43F0	0x43A0
put_pixels8_mmx	227	3.10%	393,391,000	0xFFFFFFFF	0x1F1020	0x1F2020	0x60
pix_sum16_mmx	213	2.91%	369,129,000	0xFFFFFFFF	0x332670	0x333670	0x70
MPV_decode_mb	213	2.91%	369,129,000	0xFFFFFFFF	0xAC930	0xAD930	0x2EC0
ff_fdct_sse2	172	2.35%	298,076,000	0xFFFFFFFF	0x3670A0	0x3680A0	0x500
sse16_sse2	161	2.20%	279,013,000	0xFFFFFFFF	0x332850	0x333850	0xD0

Activity ID	Activity Result	Total Samples	Duration	Machine Name
81	Fri May 08 03:31:11 2009 - Sampling Results [SAM-LAPTOP]	24735	13.328	SAM-LAPTOP

Figure 3. Hotspots by function where most time is spent

The function level information might be helpful enough, but if you want to, you can drill down into the source to see which lines of C code you spend the most time on, as shown in Figure 4 for the `diff_pixels_mmx` function, which is one of the most often called functions.



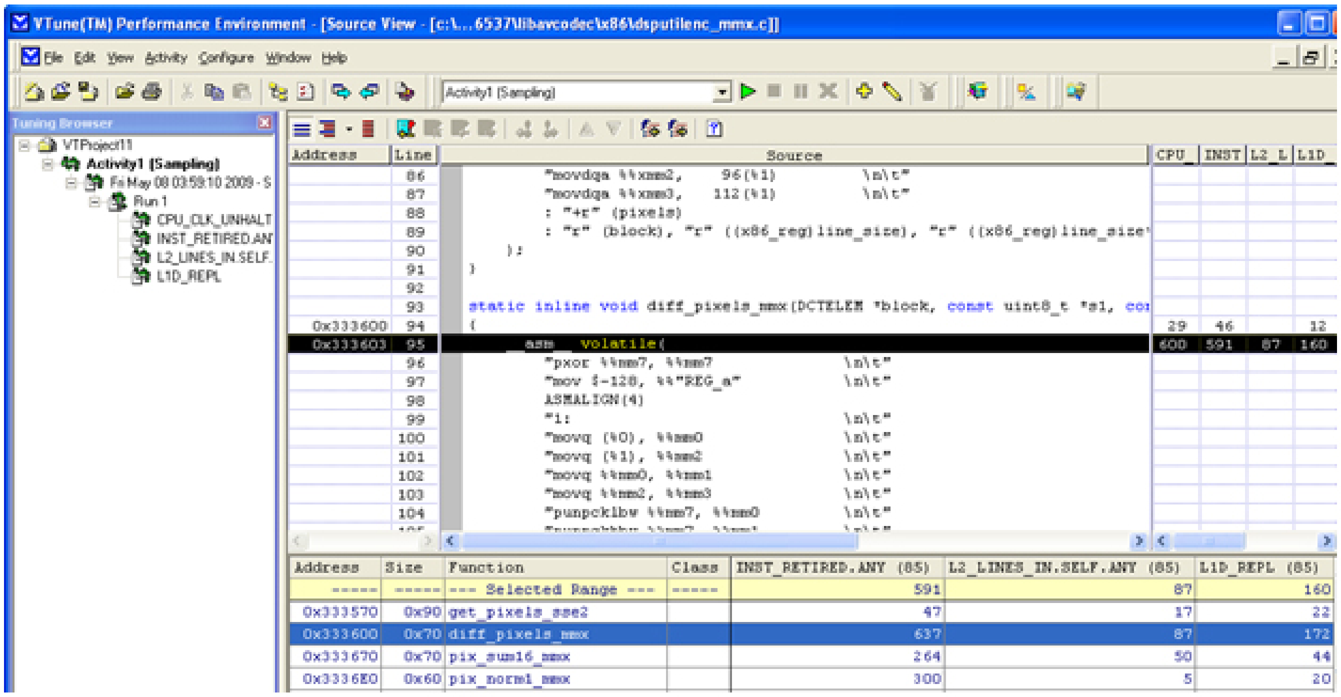
```
Address    Line  Source                                                                              CPU_
           86            "movdqa %%xmm2,    96(%1)          \n\t"
           87            "movdqa %%xmm3,   112(%1)          \n\t"
           88            : "+r" (pixels)
           89            : "r" (block), "r" ((x86_reg)line_size), "r" ((x86_reg)line_size*3)
           90        );
           91    }
           92
           93    static inline void diff_pixels_mmx(DCTELEM *block, const uint8_t *s1, const uint8_t *s2,
0x333600   94    {                                                                                   29
0x333603   95        __asm__ volatile(                                                               600
           96            "pxor %%mm7, %%mm7                \n\t"
           97            "mov $-128, %%"REG_a"             \n\t"
           98            ASMALIGN(4)
           99            "1:                               \n\t"
           100           "movq (%0), %%mm0                 \n\t"
           101           "movq (%1), %%mm2                 \n\t"
           102           "movq %%mm0, %%mm1                \n\t"
```

Address	Size	Function	Class	CPU_CLK_UNHALTED.CORE (81)
-----	-----	--- Selected Range ---	-----	600
0x333570	0x90	get_pixels_sse2		69
0x333600	0x70	diff_pixels_mmx		629
0x333670	0x70	pix_sum16_mmx		213
0x3336E0	0x60	pix_norm1_mmx		126

Figure 4. Hotspots analysis of a function

The tuning assistant suggests that your CPI (Clocks Per Instruction) might be high: You spend a larger number of clock cycles than just one per instruction on average, so that may indicate that you have hotspots where you are missing cache and stalling the processor pipeline. To determine this, you require an event-based run rather than a periodic time-based profile. While the Intel® VTune™ analyzer GUI is fairly intuitive, those less familiar with profiling methods and usage of tools like Intel® VTune™ Performance Analyzer will find the built-in Help, the examples and tutorials, and the User Manual to be quite helpful. However, the Intel white paper "Using Intel® VTune™ Performance Analyzer Events/ Ratios & Optimizing Applications (/en-us/articles/using-intel-vtune-performance-analyzer-events-ratios-optimizing-applications)" is an excellent reference that provides overview of profiling theory along with concrete examples. Here you apply the same techniques with a focus on using them to understand the specific requirements and characteristics of digital media encoding software. Once you have the hang of using Intel® VTune™ tools, give an event-based profile run a try, and see if the same `diff_pixels_mmx` function comes up as a hotspot for cache misses.

So, running again, you configured to track CPI, L1 data cache miss rate, and L2 data cache miss rate. Intel® VTune™ Performance Analyzer employs event-based counters built into the x86 architecture that are configured to generate interrupts when event counts are incremented so that the tool can sample the counts and associate events like cache misses to specific lines of code and even specific instructions. While this sounds complicated, Intel® VTune™ Performance Analyzer makes this a simple process of selection through the GUI and automatically programs the built-in PMU (Performance Monitoring Unit) registers to generate the interrupts and sampling points to profile the code being run. The cache miss rate is, in fact, found to be high with event-based sampling, as shown in Figure 5 by the L2 unified cache miss count of 87 and the L1 data miss count of 160, so it may make sense to add a prefetch for pixel data to make this code more efficient. Modifying this open source code to prefetch pixel data can be done using compiler directives. Because this case uses GCC (GNU C Compiler), refer to this compiler's directives as documented by GNU in the "Data Prefetch Support (http://gcc.gnu.org/projects/prefetch.html)" notes found on gcc.gnu.org. When pre-fetching data, carefully consider how this might affect other data in cache since cache lines must be evicted in order to make room for the pre-fetched data, and, in fact, this could cause a new cache miss hot-spot elsewhere in the code. Good background on this is provided in the GCC Data Prefetch Support notes. This would be an interesting portion of the ffmpeg code to alter with prefetch instructions or compiler directives and to re-profile to examine whether this provided an overall optimization or simply eliminated this hot-spot, but created new ones. The best way to learn how to use Intel® VTune™ tools is to start experimenting with some code changes on your own to see how they influence execution efficiency.


VTune(TM) Performance Environment - [Source View - [c:\...6537\libavcodec\x86\dsputilenc_mmx.c]]
File Edit View Activity Configure Window Help
Activity1 (Sampling)
Tuning Browser
VTProject11
Activity1 (Sampling)
Fri May 08 03:59:10 2009 - S
Run 1
CPU_CLK_UNHALT
INST_RETIRED.AN
L2_LINES_IN.SELF
L1D_REPL

```
Address   Line  Source                                                                      CPU_ INST L2_L L1D_
          86            "movdqa %%xmm2,     96(%1)          \n\t"
          87            "movdqa %%xmm3,    112(%1)          \n\t"
          88            : "+r" (pixels)
          89            : "r" (block), "r" ((x86_reg)line_size), "r" ((x86_reg)line_size'
          90        );
          91    }
          92
          93    static inline void diff_pixels_mmx(DCTELEM *block, const uint8_t *s1, co
0x333600  94    {                                                                           29   46        12
0x333603  95        __asm__ volatile(                                                       600  591  87   160
          96            "pxor %%mm7, %%mm7                  \n\t"
          97            "mov $-128, %%"REG_a"               \n\t"
          98            ASMALIGN(4)
          99            "1:                                 \n\t"
          100           "movq (%0), %%mm0                   \n\t"
          101           "movq (%1), %%mm2                   \n\t"
          102           "movq %%mm0, %%mm1                  \n\t"
          103           "movq %%mm2, %%mm3                  \n\t"
          104           "punpcklbw %%mm7, %%mm0             \n\t"
```

Address	Size	Function	Class	INST_RETIRED.ANY (85)	L2_LINES_IN.SELF.ANY (85)	L1D_REPL (85)
-----	-----	--- Selected Range ---	-----	591	87	160
0x333570	0x90	get_pixels_sse2		47	17	22
0x333600	0x70	diff_pixels_mmx		637	87	172
0x333670	0x70	pix_sum16_mmx		264	50	44
0x3336E0	0x60	pix_norm1_mmx		300	5	20

Figure 5. Confirmed cache-miss hotspot

Optimizing Digital Video Encoder Software

Video encoding software can be optimized by making sure that hotspots are executed efficiently. So, as you saw above, in a frequently executed function, you may want to modify code to try and ensure that it doesn't have cache misses and pipeline stalls by either prefetching data into cache or perhaps even locking state information into cache.

In general, you would also want to make sure that you're taking full advantage of vectorized instructions known as *Intel® Streaming SIMD Extensions* (Intel® SSE), which are instructions that vectorize instructions so that a single instruction can process multiple data words known as *Single Instruction, Multiple Data* (SIMD). The Intel® SSE instructions available with x86 processors were designed for streaming operations that would typically be found in code like ffmpeg. Much of ffmpeg does appear to use Intel® SSE and SIMD Multi-Media Extensions (MMX) to the base instruction set. You can do this can by writing assembly code and either linking it in or using inline assembly pragmas, as you saw in the `diff_pixels_mmx` function.

Alternatively, you could use the Intel® Integrated Performance Primitives (Intel® IPP) - a threaded library of functions designed for multimedia applications that is well documented in the Intel® IPP documentation (/en-us/) and a great resource are the code samples found in "Intel® IPP Sample Code (/en-us/articles/intel-integrated-performance-primitives-intel-ipp-intel-ipp-sample-code)". The Intel® IPP and use of compiler directives will help immensely with code like ffmpeg which can benefit from Intel® SSE and optimized primitives that can be used for code that is data processing intensive.

Summary

Video encoding is critical for content delivery even with gigabit networks, next-generation wireless, and broadband to the home. The need for encoding on the fly in real time is growing based on more on-demand media services, and the processor resources required is increasing based on high-bit rate, high-resolution content being downloaded and streamed more frequently to a broad range of desktop, mobile, and entertainment devices. Today's digital media production, delivery, and consumption require significant computing resources, and developers need to know how to optimize encoding/decoding software. Using Intel® VTune™ Performance Tools, without extensive training or profiling experience, we were able to quickly drill down into a hot-spot in the open source ffmpeg software to understand where most of computational complexity resides in primitive functions used for encoding. Without tools like Intel® VTune™ Performance Analyzer, it's often difficult for programmers to have a good feel for the relative cost of specific data transformation operations involved in encoding and where most of the time is being spent. You must first know where the problems are before attempting to solve them, and you need to have a tool that allows follow-up measurement after code modification to verify that code restructuring was helpful. These tools and methods are truly invaluable to those involved in digital media.

About the Author

Dr. Sam Siewert is the chief technology officer (CTO) of Atrato, Inc. and has worked as a systems and software architect in the aerospace, telecommunications, digital cable, and storage industries. He also teaches as an Adjunct Professor at the University of Colorado at Boulder in the Embedded Systems Certification Program, which he co-founded in 2000. His research interests include high-performance computing and storage systems, digital media, and embedded real-time systems.

Categories: Game Development (/en-us/search/site/language/en?query=Game%20Development) , Graphics (/en-us/search/site/language/en?query=Graphics) , Intel® VTune™ Amplifier XE (/en-us/search/site/language/en?query=Intel%C2%AE%20VTune%E2%84%A2%20Amplifier%20XE) , Desktop (/en-us/search/site/language/en?query=Desktop) , Laptop (/en-us/search/site/language/en?query=Laptop)

Tags: vcsource_type_techarticle (/en-us/search/site/field_tags/vcsource-type-techarticle-17309/language/en?query) , vcsource_domain_media (/en-us/search/site/field_tags/vcsource-domain-media-17243/language/en?query) , vcsource_os_windows (/en-us/search/site/field_tags/vcsource-os-windows-17246/language/en?query) , vcsource_platform_desktoplaptop (/en-us/search/site/field_tags/vcsource-platform-desktoplaptop-17245/language/en?query) , vcsource_product_vtunexe (/en-us/search/site/field_tags/vcsource-product-vtunexe-17902/language/en?query) , vcsource_index (/en-us/search/site/field_tags/vcsource-index-20510/language/en?query) , vcsource (/en-us/search/site/field_tags/vcsource-35148/language/en?query)

Comments (4)

^Top

said on Wed, 05/26/2010 - 07:46

i am agree with you.

said on Tue, 02/15/2011 - 22:27

I believe there is a typo in one of the commands for ffmpeg for converting to mp2. It is the last command just before Figure 1. The two commands are as follows:

```
# ffmpeg -i big_buck_bunny_720p_surround.avi -an -vcodec copy
big_buck_no_audio.mp4
# ffmpeg -i big_buck_no_audio.mp4 -vcodec mpeg2 big_buck_no_audio.mp2
```

According to what is written previously in the paper, the second line should be mpeg2video, not just mpeg2. When running the second command as is, an error is produced noting Unknown encode 'mpeg2'

said on Sun, 02/20/2011 - 18:37

Hello Justin,

Use "mpeg2video" instead of "mpeg2" in the 2nd command above. That's the ffmepg name for MP2 videos. Check other format names using ffmpeg -formats command.

-Blaze

blazevincent said on Sun, 02/20/2011 - 18:39

Hello Justin,

Use "mpeg2video" instead of "mpeg2" in the 2nd command above. That's the ffmepg name for MP2 videos. Check other format names using ffmpeg -formats command.

-Blaze

Add a Comment

Top

(For technical discussions visit our developer forums. For site or software product issues contact support.)

Please sign in to add a comment. Not a member? Join today