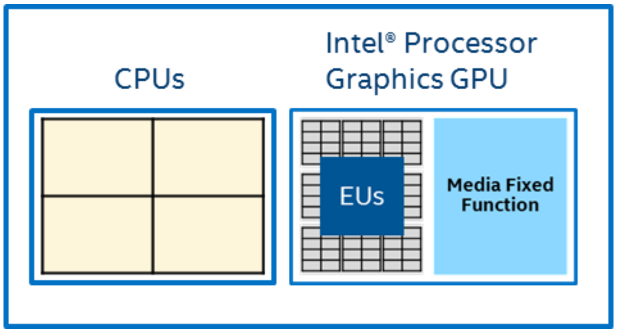
WHITE PAPER

Using Intel® VTune™ Amplifier to Optimize Media & Video Applications

Access the power of Intel® Quick Sync Video (hardware-accelerated codecs)
Visualize heterogeneous hardware operations to improve performance

Introduction

Intel® VTune™ Amplifier is an ideal software tool to help optimize your code for the amazing capabilities of Intel's hardware. This whitepaper focuses on how to use Intel® VTune™ (together with other Intel software tools) to understand performance issues in applications that access Intel® Quick Sync Video fixed function hardware via the Intel® Media SDK.



Many Intel processors contain CPU and GPU components

Where Intel® Processor Graphics is present, there are at least three different types of hardware accessible to developers:

1. SIMD CPU cores, programmed by a rich ecosystem of conventional languages.
2. General purpose EUs which can be programmed via heterogeneous languages like OpenCL™.
3. Specialized/fixed function hardware for video codecs and image processing accessible by Intel Media SDK.

In the past, it was sufficient to focus on optimizing application CPU performance. Now, especially for video processing, if your application is not using processor graphics features, some performance capabilities are untapped. Many processors across Intel's Atom™, Core™, and Xeon® product lines now have multiple components specialized for video and image processing tasks. Compartmentalizing work to fit the part of the architecture best suited for it is increasingly important since non-CPU components are a majority of total transistor count, die space, and capability for many systems. This puts more demands on developers to manage that complexity. Intel VTune Amplifier helps enable the transition to heterogeneous development by giving in-depth feedback on efficiency across all three types of hardware on the same timeline.

What is Intel® Quick Sync Video?
Intel® Quick Sync Video refers to dedicated media processing capabilities of Intel® Processor Graphics Technology. The searchable processor specification list at ark.intel.com is the authoritative site for Intel processor capabilities. Look for Processor Graphics and Intel® Quick Sync Video (available as a technology filter). The Intel® Media Server Studio site has more info on hardware support including OEMS/ODMs platforms.

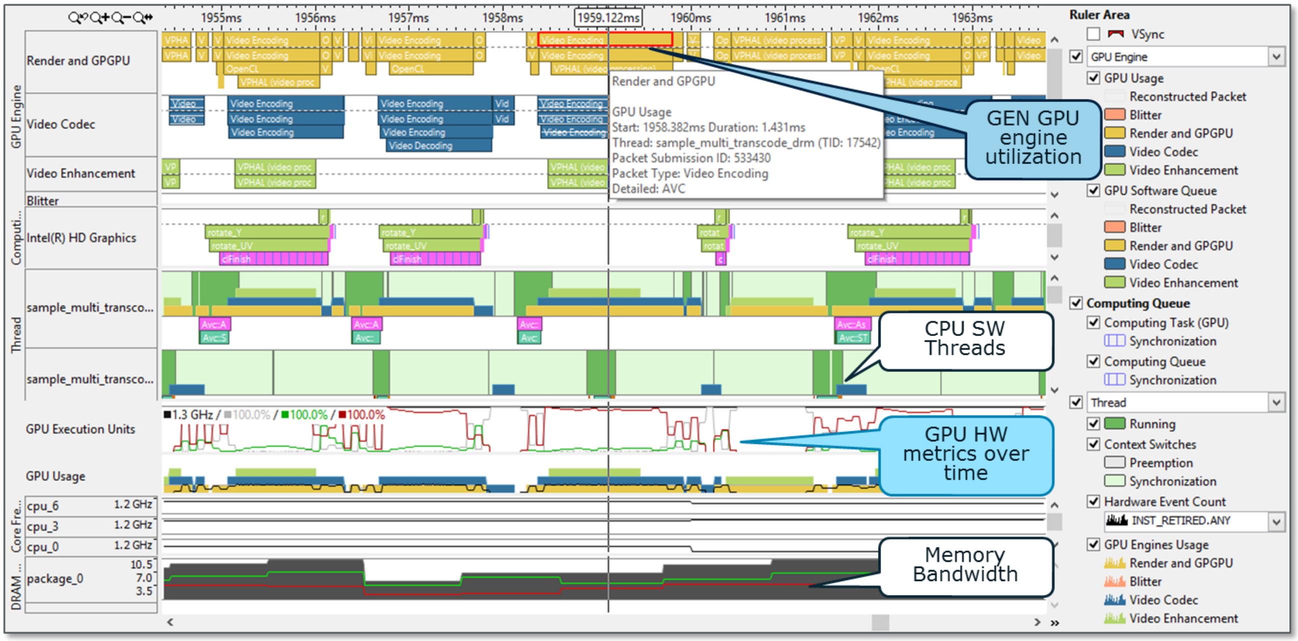
What is the Intel® Media SDK?
It is a cross-platform API that provides access to media accelerators for Intel CPU and GPUs, and is available as a free standalone tool for client applications, or bundled in the Intel® Media Server Studio for building data center and embedded media solutions and applications. The Intel Media Server Studio Professional Edition also includes Intel VTune Amplifier as part of the package.

Familiar VTune Capabilities Extended for Hardware-accelerated Media Applications

Intel VTune Amplifier is designed to let you see and understand the often complex interrelationships between heterogeneous processor components to help you get the most from Intel hardware. With enhanced capabilities to visualize CPU, execution units (EUs), and video processing fixed function utilization, the tool continues its mission of providing developers information to improve efficiency on Intel® processors in the heterogeneous era. When developing for processors with many kinds of compute engines, the ability to "see" how your application is working is more important than ever. VTune Amplifier has added ways to visualize this that are quickly intuitive and familiar to new users as well as developers familiar with the tool for CPU and/or OpenCL™ development.

As the screenshot below shows, fixed function, CPU, and EU activity can now be seen on the same timeline.

- "Render and GPGPU" shows EU activity. This can be from custom OpenCL kernels as well as prebuilt codec kernels from Media SDK. Labels are provided for easy interpretation.
- "Video Codec" shows VDbox activity. For Broadwell and newer architectures there are multiple VDboxes with their own timelines. "Video Enhancement" shows VEbox activity. This is important information for Intel Media SDK-optimized applications or applications using Intel media extensions to OpenCL.
- Other tracks show Intel GPU hardware metrics like frequency, EU stalls, etc. These metrics show how efficiently the GPU is being used – which is important for GPGPU developers using the Intel® SDK for OpenCL™ Applications (a free standalone tool for client applications, and also bundled in the free Media Server Studio Community Edition).
- Details on memory hierarchy performance and many other deep architectural insights are available with a rich set of hardware counters.



Render and GPGPU

GPU Usage
Start: 1958.382ms Duration: 1.431ms
Thread: sample_multi_transcode_drm (TID: 17542)
Packet Submission ID: 533430
Packet Type: Video Encoding
Detailed: AVC

Ruler Area
- □ VSync
- ☑ GPU Engine
  - ☑ GPU Usage
    - Reconstructed Packet
    - Blitter
    - Render and GPGPU
    - Video Codec
    - Video Enhancement
  - ☑ GPU Software Queue
    - Reconstructed Packet
    - Blitter
    - Render and GPGPU
    - Video Codec
    - Video Enhancement
- ☑ **Computing Queue**
  - ☑ Computing Task (GPU)
    - Synchronization
  - ☑ Computing Queue
    - Synchronization
- ☑ Thread
  - ☑ Running
  - ☑ Context Switches
    - Preemption
    - Synchronization
  - ☑ Hardware Event Count
    - INST_RETIRED.ANY
  - ☑ GPU Engines Usage
    - Render and GPGPU
    - Blitter
    - Video Codec
    - Video Enhancement

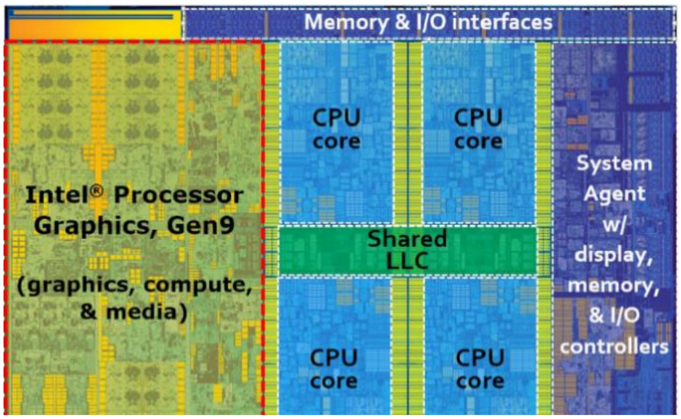
An example platform view showing heterogeneous component activity on a common timeline

The combination of performance information from all compute engines in one interface and aligned on a common timeline is powerful. You get a unified dashboard for the whole processor — a view of all components working together. This means you can see at a glance, what is working efficiently and what is not. With this information you can make your media code fast, then faster.

VTune's rich set of profiling and performance visualization features are especially important when integrating legacy CPU code with hardware accelerated codec features, which must concurrently work across all three types of compute engines. The next section provides more details on the Intel® Processor Graphics components available for video processing application development.

Intel® Processor Graphics GPU Architecture Overview

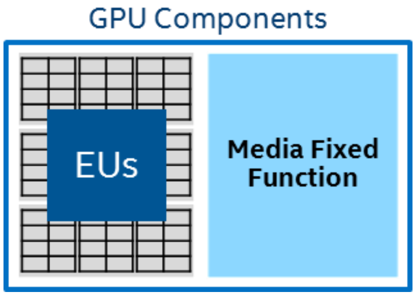
For many types of Intel processors, even a high level view of the processor die shows that the architecture is clearly heterogeneous. A significant part of the transistor count and overall capability is in the non-CPU parts of the chip, so utilizing these capabilities and visualizing efficiency there is increasingly important.


Memory & I/O interfaces
CPU core
CPU core
Intel® Processor Graphics, Gen9
(graphics, compute, & media)
Shared LLC
System Agent w/ display, memory, & I/O controllers
CPU core
CPU core

Components layout for an Intel(R) Core(TM) i7 processor 6700K

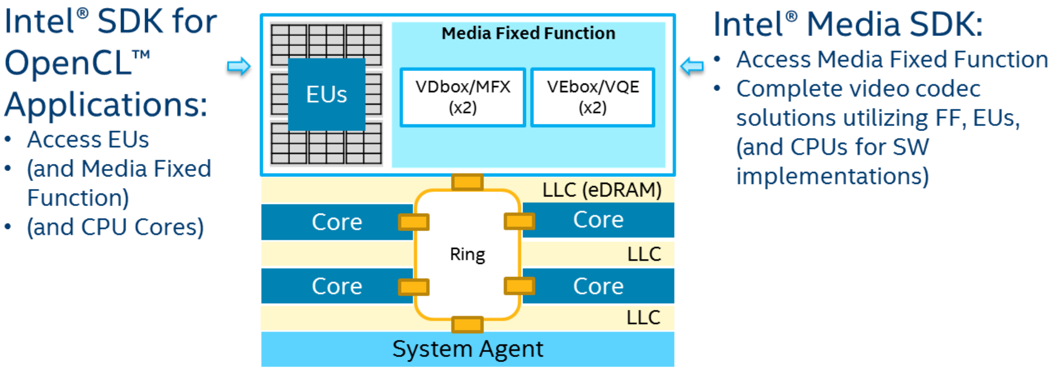
This paper focuses on VTune™ Amplifier's capabilities to visualize what is happening on the **GPU** component. (Depending on your specific target processor there may be other heterogeneous programmable components as well, which will be covered elsewhere.) The Intel GPU found in current processors is also called processor graphics or Gen graphics. Additional information is available by searching for the specific Gen architecture. For example, Intel® 6th Generation Core processors contain Gen9 architecture GPUs. Intel GPUs have two main groups of hardware of interest to media developers:

- **EUs**: General purpose execution units. These can be used for graphics rendering, but they are also suited to a wide range of media processing tasks.
- **Media Fixed Function**: In addition, specialized fixed function hardware accelerates video codec and frame processing algorithms for fundamentally higher performance at lower power than the EUs or CPUs.



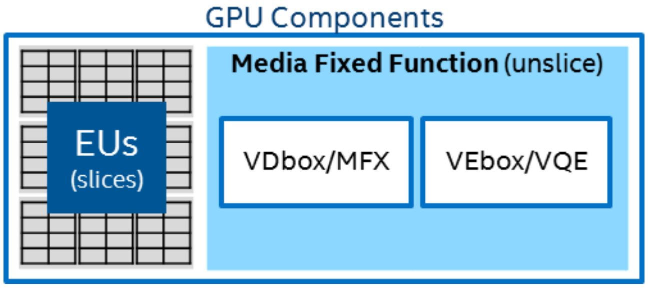
This combination means Intel GPUs have an ideal combination of features for video processing applications:

- Hardware accelerated decode and encode for many popular codecs, now including HEVC/H.265, made available through the Intel Media SDK
- Direct access to hardware components of these codecs such as motion estimation is also available for custom algorithm development via the Intel SDK for OpenCL Applications
- Hardware acceleration of many frequently used image processing algorithms (i.e. resize)
- Create custom pipelines mixing these already implemented components with your own algorithms
- High-end Intel GPUs now offer >1 TFLOPS of EU performance
- Codec operations that would require TFLOPS of CPU capability can be run simultaneously with EU and CPU tasks
- Separate clock from CPU cores (frequency can be raised or lowered independently based on workload)
- On the same ring as CPU cores and interacting with same cache hierarchy via EDRAM (available on some processor models) for low latency/high bandwidth memory transfers.


Intel® SDK for OpenCL™ Applications:
• Access EUs
• (and Media Fixed Function)
• (and CPU Cores)
EUs
Media Fixed Function
VDbox/MFX (x2)
VEbox/VQE (x2)
LLC (eDRAM)
Core
Core
Ring
LLC
Core
Core
LLC
System Agent
Intel® Media SDK:
• Access Media Fixed Function
• Complete video codec solutions utilizing FF, EUs, (and CPUs for SW implementations)

Intel CPU and EU/GPGPU features are well covered elsewhere. Please see “The Compute Architecture of Intel Processor Graphics” series for more detailed coverage of EU/GPGPU features.

In addition to EU slices, there are “unslice” components with additional hardware engines individually schedulable for media tasks. They are described in the following section to give a quick background for how they are visualized in VTune Amplifier.



VDBOX (also known as Video Box) holds the fixed function hardware blocks for encode and decode. It is also sometimes referred to as MFX (multi-format video codec). Except for the low power/latency VDenc path (in Intel Skylake processors and forward) most codecs use an ENC/PAK architecture. ENC is primarily motion search, implemented in the EUs, samplers, etc. PAK uses VDbox/MFX. They must run concurrently for best performance. MFX decode is a single operation and not split into stages.

VDbox/MFX contains:

- Bitstream decoder (BSD).
- ENC (intra prediction, motion estimation)
- PAK (quantization, entropy coding, pixel reconstruction, motion compensation)

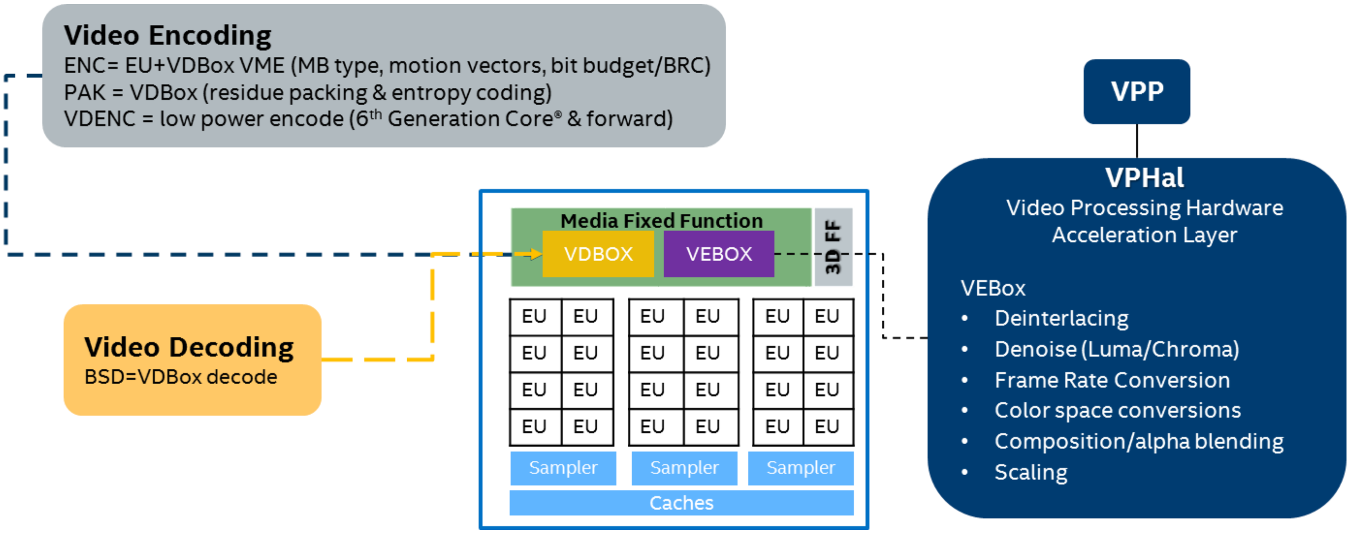
Intel® Xeon® E3-1500 v5 and 6th Generation Intel® Core processors with Iris™ Pro, Iris™ and HD Graphics introduce a new low power encode path entirely in VDBox called VDenc. However, it is lower quality.

VEBOX (video enhancement box) contains fixed function hardware for several video processing operations. The full set of standard frame processing operations also includes other GPU components like the EUs and samplers. It is also sometimes referred to as the Video Quality Engine (VQE).

VEbox/VQE contains:

- Denoise
- Advanced Deinterlace (ADI)
- Local Adaptive Contrast Enhancement (LACE)
- Camera processing features (skin tone enhancement, etc.)

Image/frame processing operations (some optimized with VEbox but may be implemented using other GPU hardware as well) are displayed in VTune Amplifier as **VPHAL** (video processing hardware acceleration layer). This abstraction layer helps ensure interface continuity between hardware generations. Because of this design it is not possible for VTune to show which filter is being used.


Video Encoding
ENC= EU+VDBox VME (MB type, motion vectors, bit budget/BRC)
PAK = VDBox (residue packing & entropy coding)
VDENC = low power encode (6th Generation Core® & forward)
Video Decoding
BSD=VDBox decode
Media Fixed Function
VDBOX
VEBOX
3D FF
EU
Sampler
Caches
VPP
VPHal
Video Processing Hardware Acceleration Layer
VEBox
• Deinterlacing
• Denoise (Luma/Chroma)
• Frame Rate Conversion
• Color space conversions
• Composition/alpha blending
• Scaling

Video processing features mapped to GPU hardware components

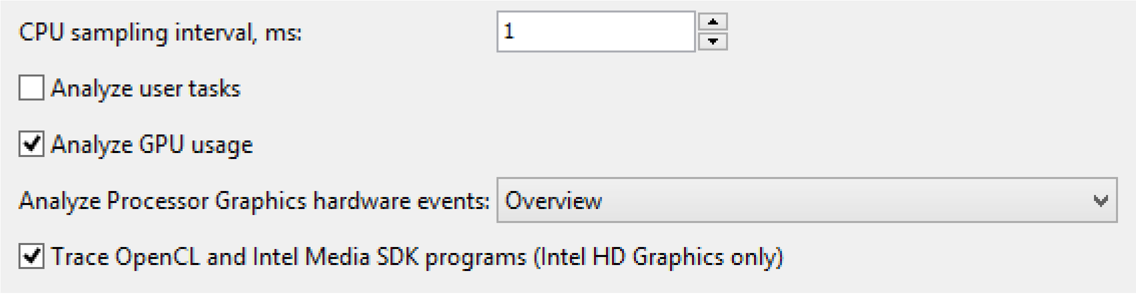
Enabling Intel Quick Sync Video Analysis in Intel VTune™ Amplifier

There are just a few steps to get started. The flow described in this section is for the Intel VTune Analyzer 2016 version, but may require even fewer steps for future versions of this tool.

For Linux*, the full set of kernel/i915 module patches and other updates from a full Intel Media Server Studio "gold" or "generic" install is required. Run as root. Set environment variables with the included script and launch.

For Windows*, GPU analysis is an exception to the usual behavior that VTune Amplifier can be run as a regular user – running as administrator is required for full GPU data collection.

Processor graphics analysis is designed to be easy. You can start with "basic hotspots" or "advanced hotspots" just like for traditional CPU analysis. There is only one additional step: select the GPU/OpenCL/Media SDK options as below. Sampling interval can be varied as needed to trade off detail vs. size.


CPU sampling interval, ms: 1

□ Analyze user tasks

☑ Analyze GPU usage

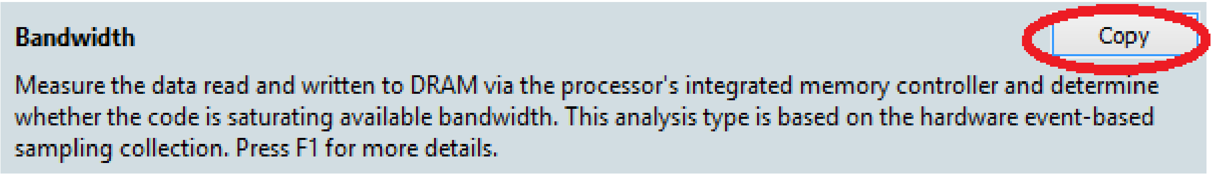
Analyze Processor Graphics hardware events: Overview

☑ Trace OpenCL and Intel Media SDK programs (Intel HD Graphics only)

Select GPU analysis options

The pre-defined hotspot collections should be fine for most use cases, and they are recommended for the first analysis passes. For more advanced scenarios you can also create custom analyses including GPU performance.

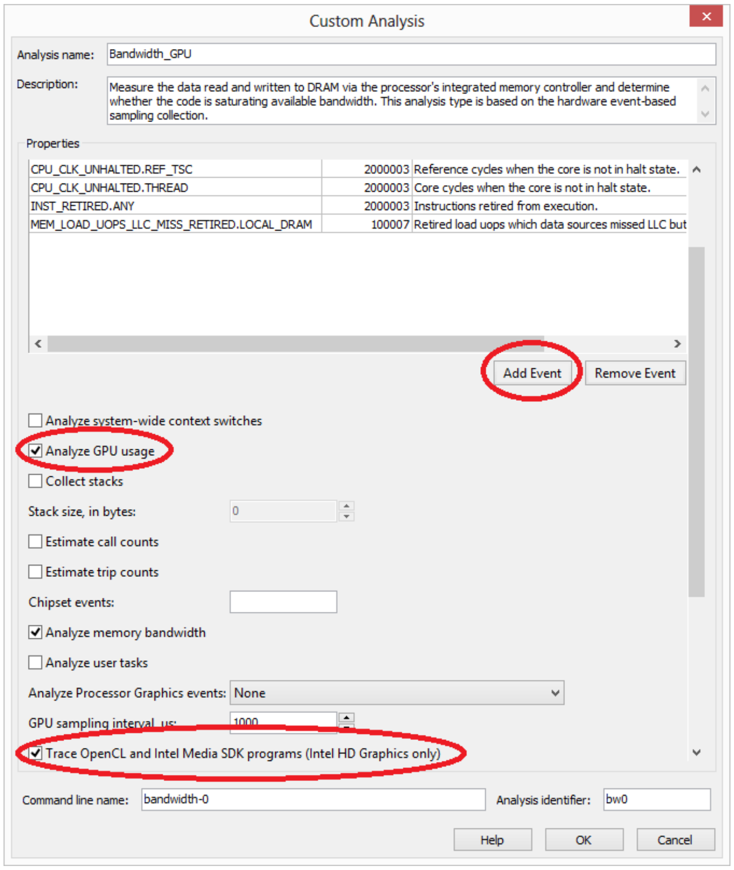
For example, to measure system memory bandwidth statistics you can 'copy' the CPU bandwidth analysis presets:



Bandwidth

Measure the data read and written to DRAM via the processor's integrated memory controller and determine whether the code is saturating available bandwidth. This analysis type is based on the hardware event-based sampling collection. Press F1 for more details.

Then extend to GPU coverage in the next screen:



Analysis name: Bandwidth_GPU

Description: Measure the data read and written to DRAM via the processor's integrated memory controller and determine whether the code is saturating available bandwidth. This analysis type is based on the hardware event-based sampling collection.

Properties

CPU_CLK_UNHALTED.REF_TSC	2000003	Reference cycles when the core is not in halt state.
CPU_CLK_UNHALTED.THREAD	2000003	Core cycles when the core is not in halt state.
INST_RETIRED.ANY	2000003	Instructions retired from execution.
MEM_LOAD_UOPS_LLC_MISS_RETIRED.LOCAL_DRAM	100007	Retired load uops which data sources missed LLC but

☐ Analyze system-wide context switches

☑ Analyze GPU usage

☐ Collect stacks

Stack size, in bytes: 0

☐ Estimate call counts

☐ Estimate trip counts

Chipset events:

☑ Analyze memory bandwidth

☐ Analyze user tasks

Analyze Processor Graphics events: None

GPU sampling interval, us: 1000

☑ Trace OpenCL and Intel Media SDK programs (Intel HD Graphics only)

Command line name: bandwidth-0

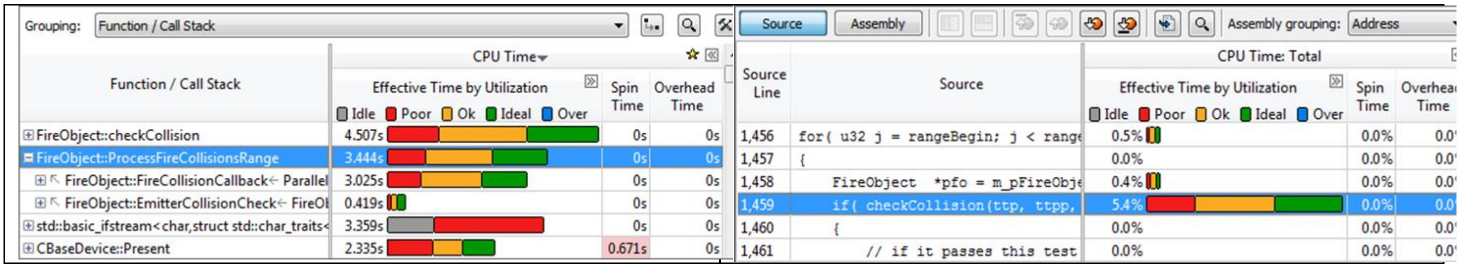
Analysis identifier: bw0

Collecting data on these memory bandwidth hardware events can give important insights, as limiting memory traffic due to extra copies is one of the most important integration issues. Please watch for more information on hardware counter selection in future updates/versions of VTune Amplifier.

Analysis Strategies for Intel Media SDK-optimized Applications

Media application analysis requires a few twists to the well-known methods for analyzing other types of programs.

For CPU and OpenCL* GPGPU applications, the usual strategy is to find hotspot code (inner loops, etc.) and make it more efficient. VTune's abilities to quickly diagnose code performance issues are quite mature. In the example below, an expensive function is quickly located (left) and code-level analysis is just a few clicks away (right panel).



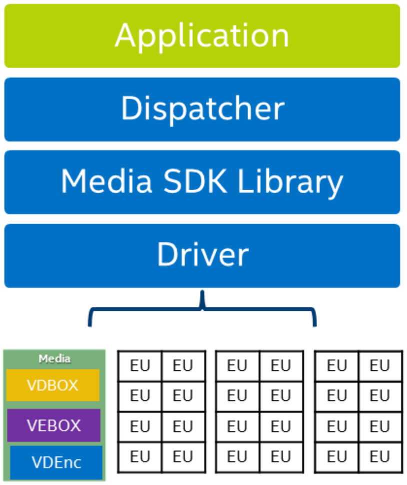
Grouping: Function / Call Stack

Function / Call Stack	CPU Time: Effective Time by Utilization (Idle, Poor, Ok, Ideal, Over)	Spin Time	Overhead Time
FireObject::checkCollision	4.507s	0s	0s
FireObject::ProcessFireCollisionsRange	3.444s	0s	0s
FireObject::FireCollisionCallback← Parallel	3.025s	0s	0s
FireObject::EmitterCollisionCheck← FireOl	0.419s	0s	0s
std::basic_ifstream<char,struct std::char_traits<	3.359s	0s	0s
CBaseDevice::Present	2.335s	0.671s	0s

Source | Assembly | Assembly grouping: Address

Source Line	Source	CPU Time: Total: Effective Time by Utilization (Idle, Poor, Ok, Ideal, Over)	Spin Time	Overhead Time
1,456	`for( u32 j = rangeBegin; j < range`	0.5%	0.0%	0.0
1,457	`{`	0.0%	0.0%	0.0
1,458	`FireObject *pfo = m_pFireObje`	0.4%	0.0%	0.0
1,459	`if( checkCollision(ttp, ttpp,`	5.4%	0.0%	0.0
1,460	`{`	0.0%	0.0%	0.0
1,461	`// if it passes this test`	0.0%	0.0%	0.0

For Intel Media SDK- optimized applications, the model is different. The goal here is to minimize gaps between Intel GPU operations and increase operation concurrency, but this can only be done indirectly. Codec operations are running via an asynchronous closed source framework working through a closed source driver. The Media SDK code in an application is therefore removed by multiple abstraction layers from what is happening on the processor.

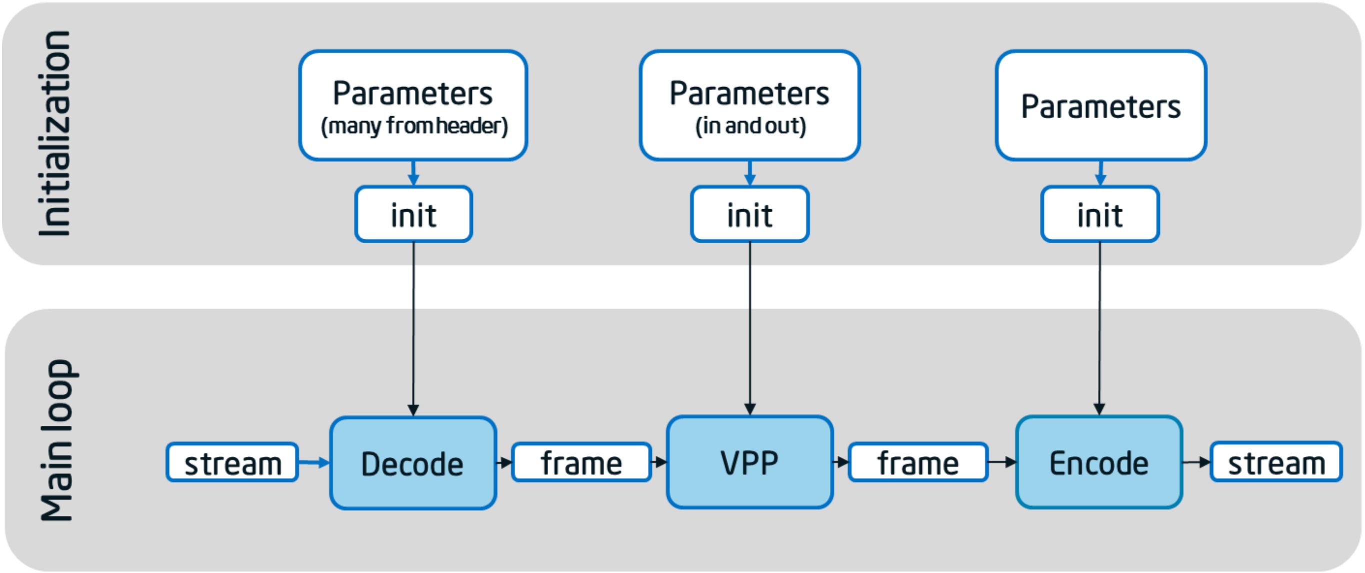


Layers between application and hardware in an Intel Media SDK programming model

Parameters are given to the pipeline components at initialization, then frames are processed asynchronously. The CPU application manages how frame surfaces are passed between stages and when operations are enqueued, but the work done on the EUs and fixed function codec hardware is otherwise independent. Since the main event collection and master timeline are aligned to the CPU perspective and symbols for the library and driver levels are not available, hotspots are harder to pinpoint. Application and Media SDK operation only align at specific sync points, usually after the last operation in the pipeline.

The Media SDK programming model's interaction with the underlying hardware is more indirect than the classic CPU model, or even GPU programming with OpenCL. For CPU applications, compiled code is executed directly on processor cores. For OpenCL applications on the EUs the application is directly involved in launching kernels, mapping/unmapping buffers, etc. There is much more direct control by the developer.

While it is certainly possible to develop completely synchronous Media SDK applications, one of the main value-adds of the Media SDK approach is that developers stand on the shoulders of giants for faster time-to-market (TTM), richer features, and the ability to start with optimal asynchronous performance out of the box. This is no small feat. Under the hood of Media SDK is a complex asynchronous implementation driven by the simple high-level interface seen by the developer.



Initialization and main loop for a Media SDK transcode

Using the pre-optimized high-level interface means that as applications move between machines (especially with different hardware generations) developers don't need to worry about implementing multiple pathways to get optimal performance. The complex scheduling underneath is already implemented and maintained by Intel, freeing developers to focus on differentiating features of their products.

Even with these limitations (required for best performance), VTune™ Amplifier can provide much valuable feedback on how efficiently your processor is functioning as a whole. If you start with an ideal application and work backwards to determine where the application being analyzed is different, this can be even more effective than locating hotspots. The Media SDK transcode implementation in *sample_multi_transcode*, one of the SDK's sample applications, is well optimized and works well for this purpose. More information on using it can be found in a later section on this topic.

There are several characteristics that all optimally functioning Intel Media SDK-optimized applications share. You can go through a checklist like the one that follows, adding improvements in iterations that allow more characteristics to be checked in your application. VTune Amplifier can be an important part of this process, showing where deviations from the ideal are happening at a much higher granularity/level of detail than through other methods.

Heuristic	
Use video memory	☑
Avoid CPU<->GPU raw frame copies	☑
Run asynchronously	☑
Minimize waits for non-GPU tasks	☑

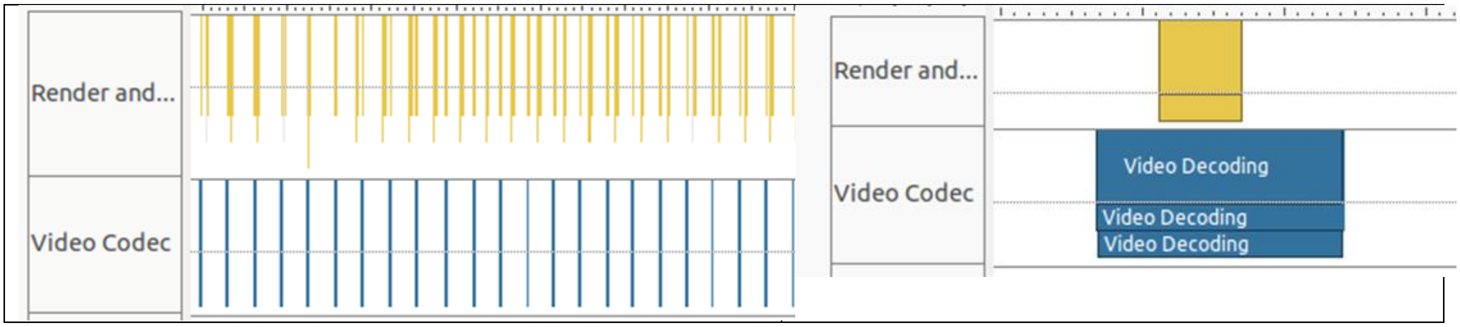
The following sections illustrate common scenarios where applications deviate from these ideal characteristics and how VTune Amplifier can be used to quickly find the largest bottlenecks.

Example 1: Minimize Waits for non-GPU Tasks

The first decode tutorial (simple_2_decode) for Intel Media SDK provides an example of an unbalanced unthreaded/synchronous application where CPU (and I/O) activity blocks efficient queuing of GPU work. This example is optimized for simplicity, not performance. Its goal is to provide the absolute minimum of code around Media SDK decode, so all it does is:

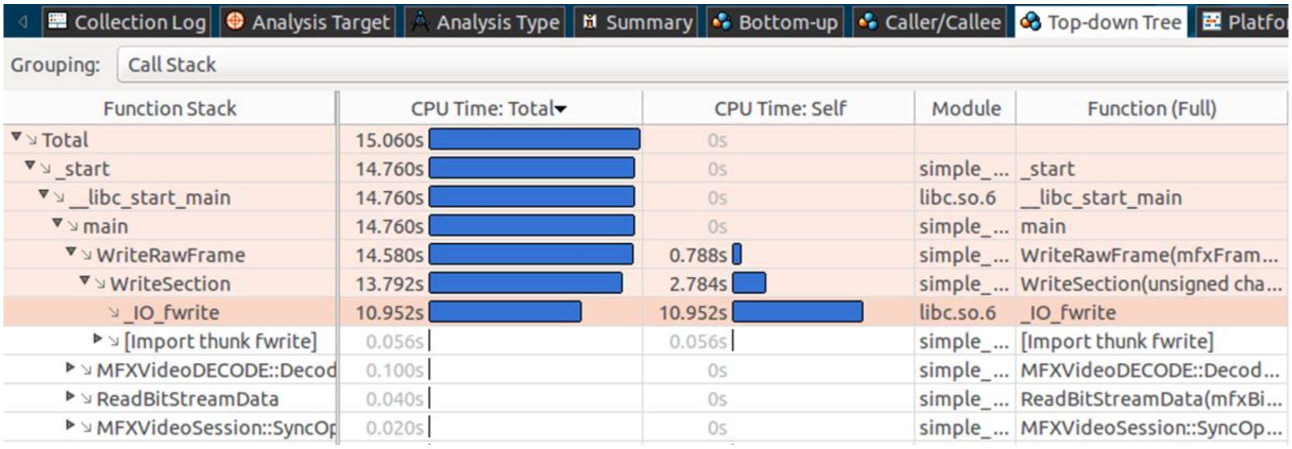
(decode)->(write to disk)->repeat

VTune quickly reveals that the GPU is not well utilized:



Fixed function decode tasks (blue) are far apart. The GPU is idle most of the time and there is little concurrency.

Looking at the CPU side, we can quickly see that something is not right. The interesting part of the application is decode, but the application run time is overwhelmingly dominated by *WriteRawFrame* – functionally a minor step. (This calls *WriteSection*, which calls standard C fwrite.)



Grouping: Call Stack

Function Stack	CPU Time: Total▾	CPU Time: Self	Module	Function (Full)
▼ Total	15.060s	0s		
▼ _start	14.760s	0s	simple_...	_start
▼ __libc_start_main	14.760s	0s	libc.so.6	__libc_start_main
▼ main	14.760s	0s	simple_...	main
▼ WriteRawFrame	14.580s	0.788s	simple_...	WriteRawFrame(mfxFram...
▼ WriteSection	13.792s	2.784s	simple_...	WriteSection(unsigned cha...
_IO_fwrite	10.952s	10.952s	libc.so.6	_IO_fwrite
▶ [Import thunk fwrite]	0.056s	0.056s	simple_...	[Import thunk fwrite]
▶ MFXVideoDECODE::Decod	0.100s	0s	simple_...	MFXVideoDECODE::Decod...
▶ ReadBitStreamData	0.040s	0s	simple_...	ReadBitStreamData(mfxBi...
▶ MFXVideoSession::SyncOp	0.020s	0s	simple_...	MFXVideoSession::SyncOp...

The *WriteRawFrame* function's only job is to write the output of Media SDK decode to disk. Since it is running on the CPU side, VTune allows quickly clicking down to the code. First *WriteSection*:

```
mfxStatus WriteSection(mfxU8* plane, mfxU16 factor, mfxU16 chunksize,
            mfxFrameInfo* pInfo, mfxFrameData* pData, mfxU32 i,
            mfxU32 j, FILE* fSink)
{
  if (chunksize !=
    fwrite(plane +
        (pInfo->CropY * pData->Pitch / factor + pInfo->CropX) +
        i * pData->Pitch + j, 1, chunksize, fSink))
    return MFX_ERR_UNDEFINED_BEHAVIOR;
  return MFX_ERR_NONE;
}
```

This function writes chunksize bytes. This could be many (as in the case where the luma plane is written one line at a time) or just 1 byte at a time as is the case with the chroma planes. For YUV 4:2:0 subsampling the amount of data in the chroma planes is ¼ of the total resolution, so this is a relatively insignificant operation in terms total output. VTune lets us quickly see that the single byte writes for chroma are especially costly, explaining why a step - which might be overlooked without the tool - considered as trivial, actually burns up most of the time.

```
mfxStatus WriteRawFrame(mfxFrameSurface1* pSurface, FILE* fSink)
{
  mfxFrameInfo* pInfo = &pSurface->Info;
  mfxFrameData* pData = &pSurface->Data;
  mfxU32 i, j, h, w;
  mfxStatus sts = MFX_ERR_NONE;

  // write Y (luma) plane
  for (i = 0; i < pInfo->CropH; i++)
    sts =
      WriteSection(pData->Y, 1, pInfo->CropW, pInfo, pData, i, 0,
             fSink);

  // write UV (chroma) planes
  h = pInfo->CropH / 2;
  w = pInfo->CropW;
  for (i = 0; i < h; i++)
    for (j = 0; j < w; j += 2)
      sts =

        WriteSection(pData->UV, 2, 1, pInfo, pData, i, j,
               fSink);
  for (i = 0; i < h; i++)
    for (j = 1; j < w; j += 2)
      sts =

        WriteSection(pData->UV, 2, 1, pInfo, pData, i, j,
               fSink);
  return sts;
}
```

The reason for the single byte writes is to do a color format conversion with minimal code. Intel Quick Sync Video hardware decode and encode works with NV12. This has a luma (Y) plane followed by interleaved chroma (UVUVUV). Many utilities for working with raw frames default to I420 format, which is a Y plane followed by separate U and V planes. So, to output U and V the same buffer is passed over twice with different offsets:

U		U		U	

->UUU...

	V		V		V

->VVV...

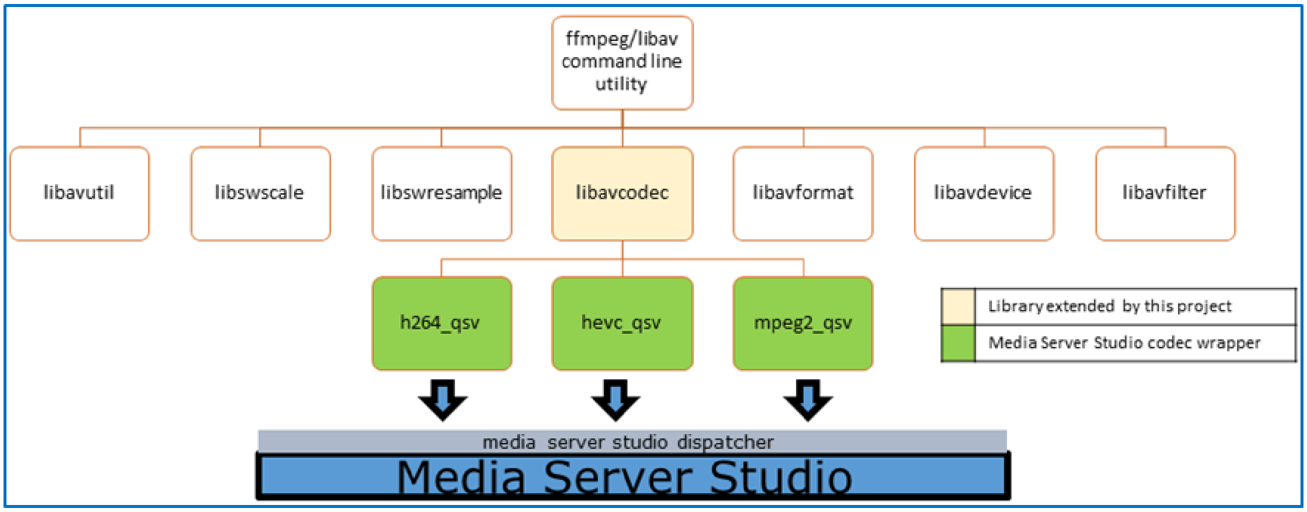
The quick (but incomplete) solution would be to optimize away the obvious inefficiencies at this hotspot. The first step would be to convert frames to I420 in memory, which would be significantly faster. Many optimized NV12->I420 conversion functions exist, including in Intel's IPP. Writing out an entire frame at a time would allow the I/O infrastructure to be much more efficient. Even better, video memory could be used and the color conversion could be done on the GPU.

However, an even better approach would be to consider the full end-to-end solution. While it is common to think of only optimizing hotspots, moving toward a holistic solution by rethinking the problem from the perspective of how the underlying hardware works can lead to a better overall approach. In this case, benchmarking decode separately makes sense from a CPU implementation perspective, but not from a heterogeneous GPU perspective. For example, if the full use scenario is transcode an efficient Gen GPU implementation would set up concurrent encodes and decodes. On Intel processor graphics hardware, transcode is faster than running encode or decode separately. With transcode being the overhead of additional synchronization and moving raw frames to/from the CPU can be avoided. The efficient hardware-focused system perspective provided by VTune guides the analysis in the right direction.

The purpose of this section is not simply to show how decodes can be optimized. This is just one example (with easily accessible source code) of a bigger class of issues where CPU and GPU work aren't well balanced. As the balance improves, VTune can visualize how GPU efficiency increases by showing smaller gaps. This is especially important for GPU operations — if gaps are too large, then GPU frequency may drop due to low utilization, reducing GPU performance.

Example 2: Copies

Starting with FFmpeg 2.8, wrappers for Intel Media SDK codecs have been available as h264_qsv, hevc_qsv, and mpeg2_qsv. At the time of writing this paper, few optimizations were added to the minimal glue code implementation of the standard libavcodec interface. Here, the Media SDK code was added underneath the synchronous "process one frame" interface.


ffmpeg/libav command line utility
libavutil
libswscale
libswresample
libavcodec
libavformat
libavdevice
libavfilter
h264_qsv
hevc_qsv
mpeg2_qsv
Library extended by this project
Media Server Studio codec wrapper
media server studio dispatcher
Media Server Studio

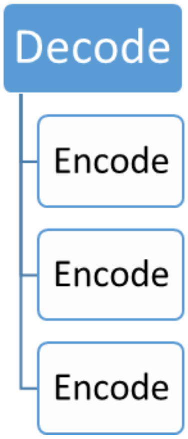
Where Media SDK "glue" code was added to FFmpeg

Using Media SDK "under" FFmpeg like this is expected to be popular because it makes hardware acceleration easy – just switch the codec name. It is also representative of a common class of integration issues.

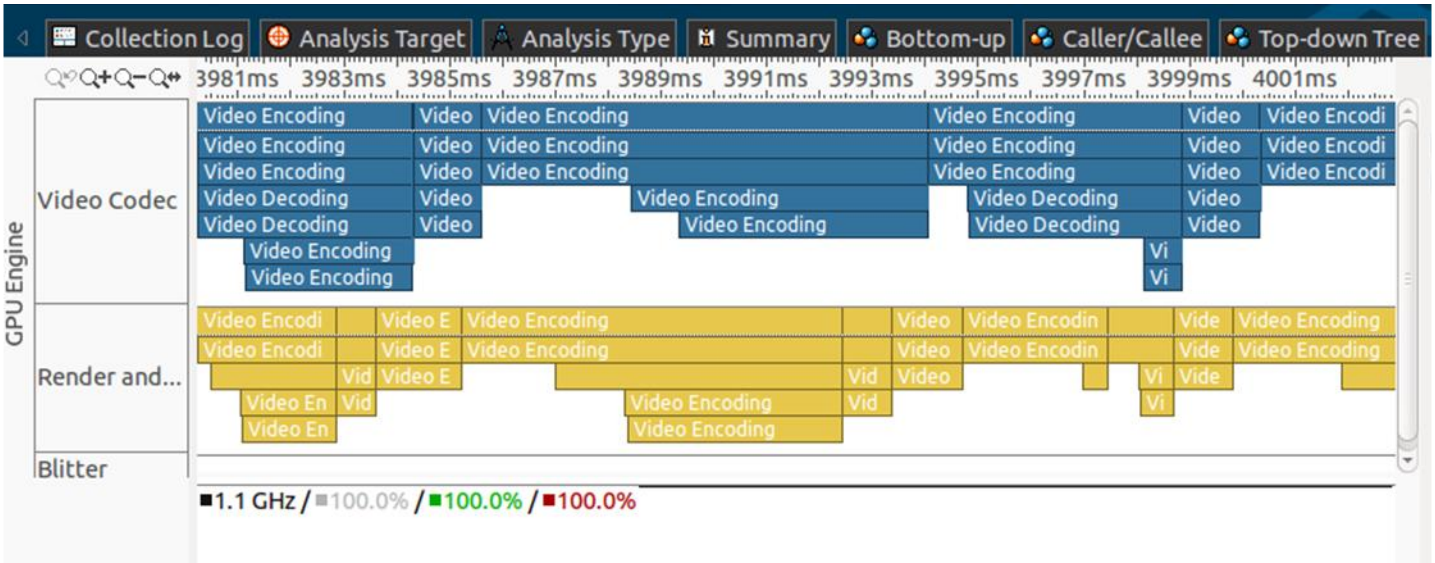
While video elementary stream decode/encode is often the main target for optimizing media applications, this is only a small component of what most media applications need to do. Container handling, audio, network protocols, etc., are necessary to build a full application. However, when there are major architecture differences between components (i.e. synchronous vs. asynchronous), copies are often the first pass solution to plug together a solution and prove it works even though they introduce inefficiencies.

The command line info that follows is provided to simplify comparison with an ideal implementation in *sample_multi_transcode*. A single decode can provide frames for multiple pipelines. This is usually to prepare a single input for playback on a variety of devices with multiple resolutions, bitrates, etc.

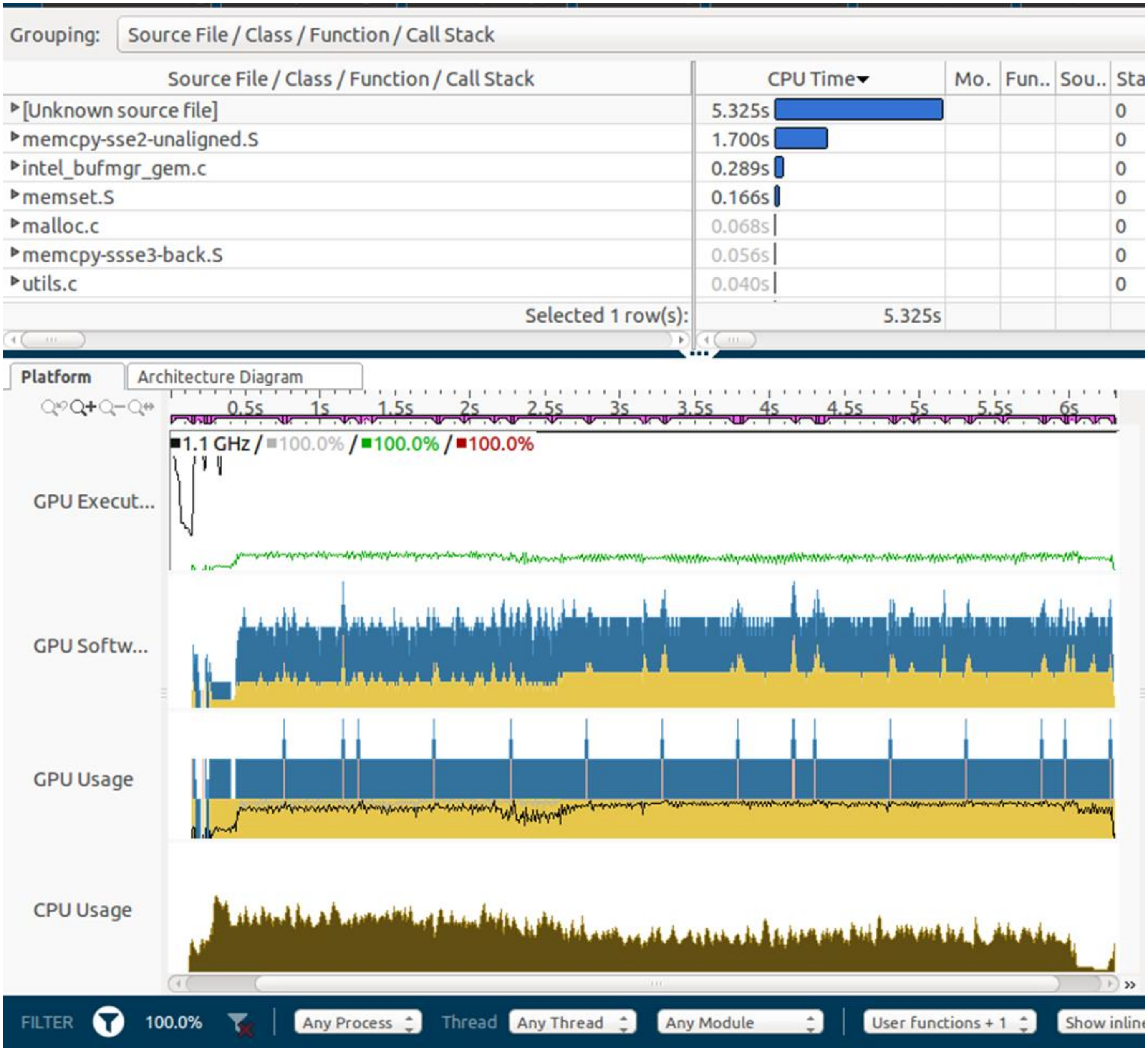
Example h264_qsv command line

h264 tests are 1:N:  	This specifies one input (-i) and several parameter sequences ending with an output. Usually other steps like resize would be included, but for simplicity, the performance tests for this paper specify multiple outputs at the same bitrate, and with the same resolution as the input. ffmpeg -y -i infile \\ -vframes 2000 -b:v 8000K -vcodec h264_qsv -preset veryfast out00.h264 \\ -vframes 2000 -b:v 8000K -vcodec h264_qsv -preset veryfast out01.h264\\ -vframes 2000 -b:v 8000K -vcodec h264_qsv -preset veryfast out02.h264

A VTune analysis of this pipeline shows lots of concurrent work with few gaps. Without comparing with *sample_multi_transcode* (next section), it may not immediately seem like there is anything to improve.



However, further analysis shows that memory copy time is much larger than expected. VTune shows that *'memcpy-sse2-unaligned'* is one of the top time uses. The CPU use, while lower than the 100% expected for pure software codec implementations, is unexpectedly high considering that we should have GPU offload.



Grouping: Source File / Class / Function / Call Stack

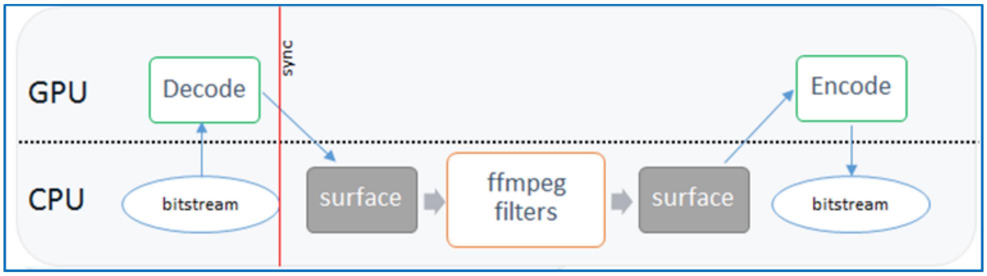
Source File / Class / Function / Call Stack	CPU Time▾	Mo.	Fun..	Sou..	Sta
▸[Unknown source file]	5.325s				0
▸memcpy-sse2-unaligned.S	1.700s				0
▸intel_bufmgr_gem.c	0.289s				0
▸memset.S	0.166s				0
▸malloc.c	0.068s				0
▸memcpy-ssse3-back.S	0.056s				0
▸utils.c	0.040s				0
Selected 1 row(s):	5.325s				

Memory bandwidth analysis (not shown) indicates that memory traffic is highly correlated to the CPU usage timeline above. Excessive CPU<->GPU copies increase CPU utilization. While Media SDK's internal copies from video to system memory are a small contributor, the main source of copy overhead is additional copies done by the FFmpeg framework.

The copies are not normally a major bottleneck with CPU encode. With decode and encode "optimized away," the copies become a much more significant contributor to overall runtime for the Media SDK wrapper *h264_qsv codec*.

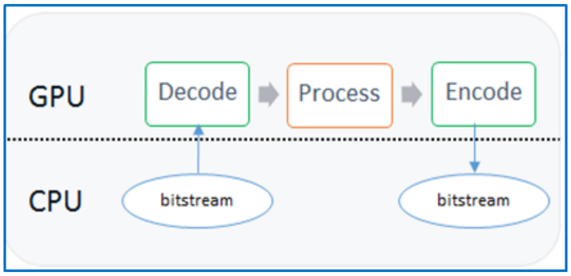
The FFmpeg implementation is constantly improving and copy overhead can be reduced. However, there are some limitations to thinking only in terms of an "accelerator" offload of hotspot operations, instead of an end-to-end solution reflecting the hardware.

In this case, the FFmpeg framework assumes serial operation. For Media SDK, this means extra synchronizations, and using single decode and encode stages as in the below diagram.



GPU/CPU timeline for h264_qsv

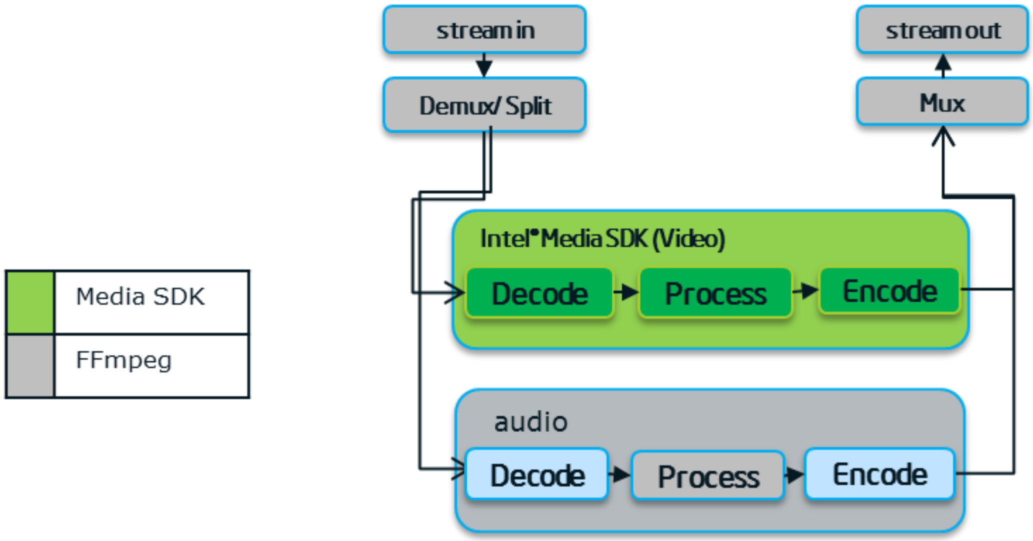
The ideal case for Media SDK performance would put full pipelines on the GPU, which allows the application to use more approaches from the ideal checklist: video memory, avoid copies, run asynchronously, with the minimum dependencies between CPU and GPU. This is true in the case of FFmpeg, as well as many other integration projects.



Ideal Media SDK pipeline

An ideal solution could be achieved by using FFmpeg for audio and I/O: containers, network protocols, etc. However, instead of using Media SDK “under” the FFmpeg framework, video packets can be taken from the splitter and fed to a specialized Media SDK pipeline. Bitstream output can be fed to FFmpeg's muxer APIs.

For more information on this approach, see: *Intel Integrating Intel® Media SDK with FFmpeg for mux/demuxing and audio encode/decode usages*.



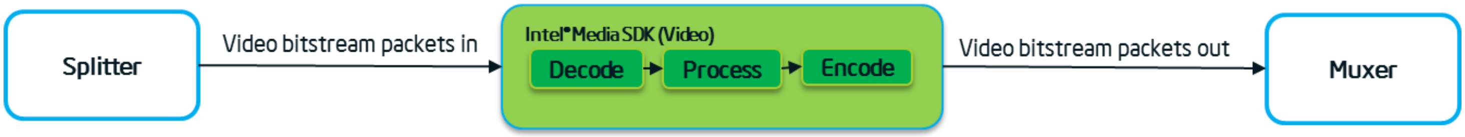
Best of both: Media SDK asynchronous pipeline integrated with FFmpeg container/audio

Other integrations can use a similar approach. Start simply, with inefficient synchronous offloads of single operations for single frames and extra copies. This allows a quick start and a path to better performance through incremental improvements. As with hotspot analysis for CPU/OpenCL performance, VTune Amplifier makes it easy to find the top inefficiency to fix next, as well as when there will be diminishing returns for optimization efforts.

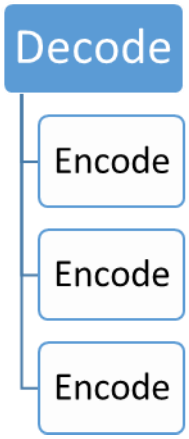
Comparing with *sample_multi_transcode*, an Ideal Implementation

Because VTune Amplifier analysis for Intel Media SDK applications is less direct than for CPU/OpenCL apps, visualizing the final goal is especially important. Many transcode scenarios can be simulated with the *sample_multi_transcode* example for Media SDK.

For a pipeline like below, instead of using a full featured framework to construct the pipeline file, reads and writes could simulate feeding bitstream packets in from the splitter or passing them to the muxer.



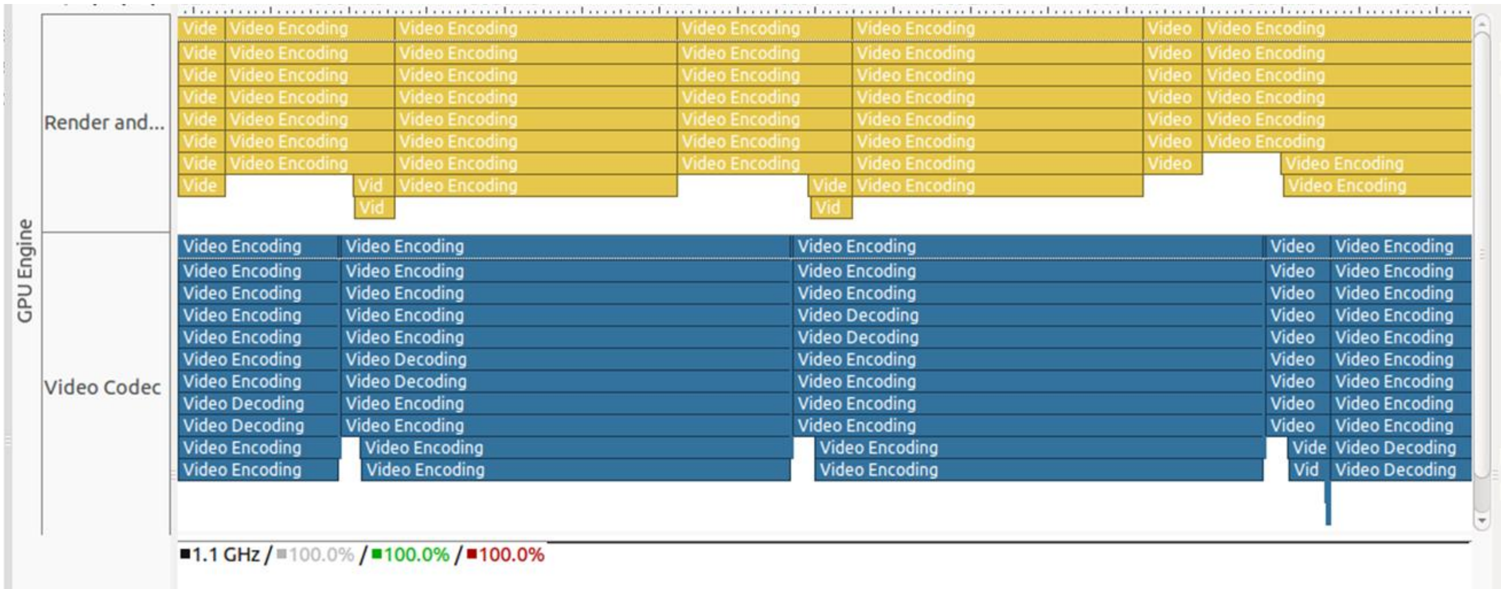
Once files are prepared, par files can be prepared for elementary stream I/O. The example that follows is for a 1:N transcode matching the FFmpeg command line in the previous section.

h264 tests are 1:N:  	sample_multi_transcode tests can be set up using the "par files" (files with parameters) using the sink/source syntax:. example par file: -i::h264 in.264 -o::sink -i::source -b 8000 -tu 7 o::h264 out00.h264 -i::source -b 8000 -tu 7 o::h264 out01.h264 -i::source -b 8000 -tu 7 -o::h264 out02.h264

Tests run with *sample_multi_transcode* will show how much performance is being left on the table, which can help determine ROI for additional work.

Knowing when to stop optimizing is important too.

For comparison, the test run earlier with FFmpeg looks like the graph below with *sample_multi_transcode*. While in both cases concurrent work is happening, there are no gaps, etc., this test shows just how much more performance is available. Note that there is significantly more concurrency, correlating with the ~2x improvement in performance vs. the FFmpeg implementation.


GPU Engine
Render and...
Vide Video Encoding Video Encoding Video Encoding Video Encoding Video Video Encoding
Vide Video Encoding Video Encoding Video Encoding Video Encoding Video Video Encoding
Vide Video Encoding Video Encoding Video Encoding Video Encoding Video Video Encoding
Vide Video Encoding Video Encoding Video Encoding Video Encoding Video Video Encoding
Vide Video Encoding Video Encoding Video Encoding Video Encoding Video Video Encoding
Vide Video Encoding Video Encoding Video Encoding Video Encoding Video Video Encoding
Vide Video Encoding Video Encoding Video Encoding Video Encoding Video Video Encoding
Vide Vid Video Encoding Vide Video Encoding Video Encoding
Vid Vid
Video Codec
Video Encoding Video Encoding Video Encoding Video Video Encoding
Video Encoding Video Encoding Video Encoding Video Video Encoding
Video Encoding Video Encoding Video Encoding Video Video Encoding
Video Encoding Video Encoding Video Decoding Video Video Encoding
Video Encoding Video Encoding Video Decoding Video Video Encoding
Video Encoding Video Decoding Video Encoding Video Video Encoding
Video Encoding Video Decoding Video Encoding Video Video Encoding
Video Decoding Video Encoding Video Encoding Video Video Encoding
Video Decoding Video Encoding Video Encoding Video Video Encoding
Video Encoding Video Encoding Video Encoding Vide Video Decoding
Video Encoding Video Encoding Video Encoding Vid Video Decoding
■1.1 GHz / ■100.0% / ■100.0% / ■100.0%

Conclusion

Intel® Quick Sync Video provides compelling performance for today's and tomorrow's rapidly increasing media workloads. Intel® VTune™ Amplifier can help identify how to make full use of the heterogeneous capabilities of Intel® Processor Graphics GPU technology when using Intel® Media SDK.

Use VTune to:

- Optimize the whole platform (Intel CPU and GPU together)
- Understand and visualize concurrent activity across hardware blocks
- See inefficiencies quickly
- See progress as your application moves closer to an ideal implementation like *sample_multi_transcode*

More Resources

The Compute Architecture of Intel® Processor Graphics Gen9

Intel Media Server Studio (for server and embedded media solution and applications)

- All editions include the Intel Media SDK and Intel® SDK for OpenCL™ Applications; the Community Edition is free
- The Professional Edition also includes Intel® VTune™ Amplifier

Intel Media SDK (free standalone tool for client applications)
Intel SDK for OpenCL™ Applications (free standalone tool for client applications)

Intel® VTune™ Amplifier (available also as a standalone tool)

Intel VTune™ Amplifier: Getting started with OpenCL™ performance analysis on Intel® HD Graphics

Intel Quick Sync Video Technology on Intel® Iris™ Graphics and Intel® HD Graphics family – Flexible Transcode Performance and Quality

Get Amazing Intel GPU Acceleration for Media Pipelines - Webinar Replay

01.org/linuxgraphics/documentation/hardware-specification-prms

Notices

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 complete information visit http://www.intel.com/performance.

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

This paper is for informational purposes only. THIS DOCUMENT IS PROVIDED “AS IS” WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION, OR SAMPLE. Intel disclaims all liability, including liability for infringement of any property rights, relating to use of this information. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted herein.