White Paper

Performance Interactions of OpenCL* Code and Intel® Quick Sync Video on Intel® HD Graphics 4000

Introduction

Developers of video editing and other applications that generate or process frames of video, and subsequently encode them using Intel® Quick Sync Video, may find it challenging to gain performance advantages using OpenCL* to shift frame processing from the CPU to Intel® HD Graphics. This paper explains why this is and how to identify workloads that can show positive improvement, enabling developers to make the most productive use of their OpenCL coding efforts.

Contents

An Unexpected Problem

In support of preparing OpenCL on Intel processor graphics for the 3rd generation Intel® Core™ processor launch, I optimized a number of OpenCL kernels for a video editing application. The expectation was that by moving some processing of video image effects from the CPU to processor graphics, application performance would be substantially improved.

The OpenCL kernels did indeed show good unit test performance gains as compared to a single CPU core running the same effect on well-optimized code—on average about three times (3x) faster, with some kernels as much as 20x faster than the equivalent CPU code! This seemed to bode well for performance gains. And in fact, after integrating the OpenCL kernels into the application, some workloads did show improvement.

But on many workloads, the same kernels that were doing well in other workloads not only failed to provide gains, but often ran slower than the CPU-coded equivalent.

What was going on?

A Bit of Background

The video editing application in question was very well coded to take good advantage of processor parallelism. The video effects were coded with SIMD instructions, and (on a four core CPU) four threads were used to independently process each video frame. The application also made use of Intel Quick Sync Video to accelerate video decoding prior to applying video effects, and to re-encode the resulting video after.

To run OpenCL as quickly as possible, one out of four threads would typically be dedicated to launching effect kernels on the GPU using OpenCL, reducing the number of threads processing effects on the CPU by the same number. This was judged an acceptable trade-off, since the GPU OpenCL kernels typically processed substantially faster than one CPU thread, when tested independently. The lost performance from dropping one CPU thread was expected to be more than recovered by gains from the OpenCL GPU threads.

However, it was discovered that performance did not improve and often decreased by moving effect processing from CPU threads to GPU threads using OpenCL.

About the developers tools used

To be able to isolate the problem and to optimize the application, I used Intel® Visual Computing Developer Tools available as free downloads at the Intel® Visual Computing Source. Intel® Media SDK was used for Intel Quick Sync Video hardware accelerated video encoding.

The Intel® SDK for OpenCL* Applications was used for OpenCL application development. For media and graphics optimization, I used the Intel® Graphics Performance Analyzers (Intel® GPA).

Isolating the Issue

First, it was noted that the most obvious difference between workloads that showed OpenCL benefits and the workload that showed performance degradation was that video in the latter was being encoded to AVC video (H.264/MPEG4 Part 10) using Intel Quick Sync Video. It was also noted that if Intel® Quick Sync Video was not used, application performance was substantially lower, but moving some effect processing from CPU threads to the GPU would increase performance in that case—though not to an extent that exceeded performance with Intel Quick Sync Video.

The next step was to apply some of Intel's performance analysis tools to get a better idea of what was happening. The Intel Graphics Performance Analyzers (Intel GPA) include a Performance Monitor that can be used to capture traces of programmable graphics processing. Those traces can then be examined in the Intel® GPA Platform Analyzer, yielding charts of processing on the GPU over time—with bars whose number and widths give a rough indication of the amount and duration of intervals of GPU processing. Areas of the bars can also be selected to obtain precise counts of total time and processing time over the selected region to derive a fairly precise measure of GPU utilization.

Figure 1 shows the integrated processor graphics activity over about two seconds, of a typical workload with a well-optimized video effect running on 4 cores of the CPU, and video encode to AVC (H.264 part 10) also executing on the CPU—no OpenCL or Intel Quick Sync Video.

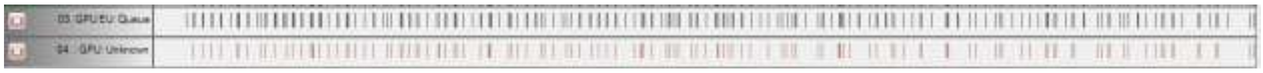

Figure 1 – Intel GPA Performance Monitor trace – no work added to GPU

This shows a very small amount of GPGPU (general purpose processing on graphics processing units) processing (measured under 1% of utilization)—likely due to updating small amounts of on-screen graphic animation (a progress bar, etc.).

The next figure shows the same workload but using Intel Quick Synch Video for all AVC encoding. The row of "GPU Unknown" remains about the same, with three new rows related to processing the video encode. The row labeled "04: GPU: ENCODE" is executing on the integrated graphics execution units, as is the row labeled "GPU: VPP", while the last row labeled "06: GPU: ENCODE" represents encode processing on dedicated function hardware such as hardware motion estimation, not the GPGPU.

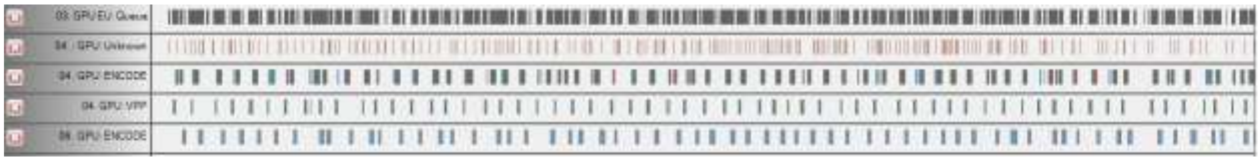

Figure 2 – Intel GPA trace with Quick Synch Video encode

The main thing to note here is that the video encode does not take up all of the processor graphics execution time. That is, if all the bars representing GPGPU execution time were collapsed into a single row, they would only fill about 40% of the total execution time—the amount of active GPU execution time measured with the Intel GPA tool. The "GPU EU Queue" row shows time when there are tasks queued to run on the GPU, but are not yet executing because the GPU is already busy. Gaps indicate that the task queue has run out of work to do on the GPU.

So there should be time remaining to process video effects on the programmable graphics execution units, and so perhaps OpenCL on Intel processor graphics can increase overall throughput.

The next figure again shows two seconds of the same workload, still using Intel Quick Sync Video for all video encoding, but also using OpenCL on the GPU to implement the video effect. Three other CPU threads continue to process the video effect, but one CPU thread (out of 4) is now moved to running the video effect on the GPU, which shows up as the new second row with heavy activity on the GPGPU.

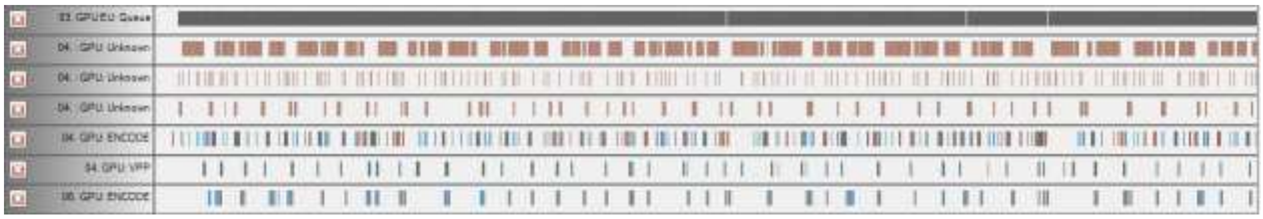

Figure 3 – Trace with OpenCL effect processing added

Note that now the top row, the GPU EU Queue, is now nearly solid, indicating that there are almost always tasks waiting to execute, because the GPGPU is nearly fully occupied. Less easy to see, is the fact that if all the rows of the GPU: Unknown, GPU: Encode and GPU: VPP were combined, they would also create a nearly solid bar, indicating nearly complete filling of the GPGPU execution units. In fact, measurements suing Intel GPA showed about 95% GPGPU utilization—a 55% increase. So it appears that we have success—nearly full use of the GPU processing capacity. (Some GPGPU execution is lost to overhead getting threads started on the GPU.)

However, timing of this application workload showed poorer actual performance with a run time about 20% longer! The last row of the Figure 3 (GPU: ENCODE) as compared to Figure 2 also shows significantly fewer bars representing units of video frame encode. This illustrates the lower frame throughput rate, since all frames are encoded on the GPU. (Note that it will often be possible to determine the frame rate from the application code, which can be combined with the GPU utilization shown by Intel GPA to verify the average per-frame video encode processing time.)

What Happened?

The most obvious possible explanation is that processing the video effect algorithm on the GPU was slower than running it on the CPU. However, testing the video effect separately indicated that the particular effect illustrated above runs about 2x faster (using all of the GPGPU execution units) than one CPU thread. So it still seems the GPU version should be faster than the CPU thread replaced.

However, as noted above and illustrated in Figure 4, Intel Quick Sync Video takes up about 40% of the GPGPU execution time—10% for each of the four CPU threads processing effects. The effect processing on the GPGPU adds 55% more to that, meaning the GPGPU effects processing must be about twice as fast just to have a chance at matching the original CPU thread performance. (Moving video encode to the CPU would be even slower overall, so that is not really an option.)

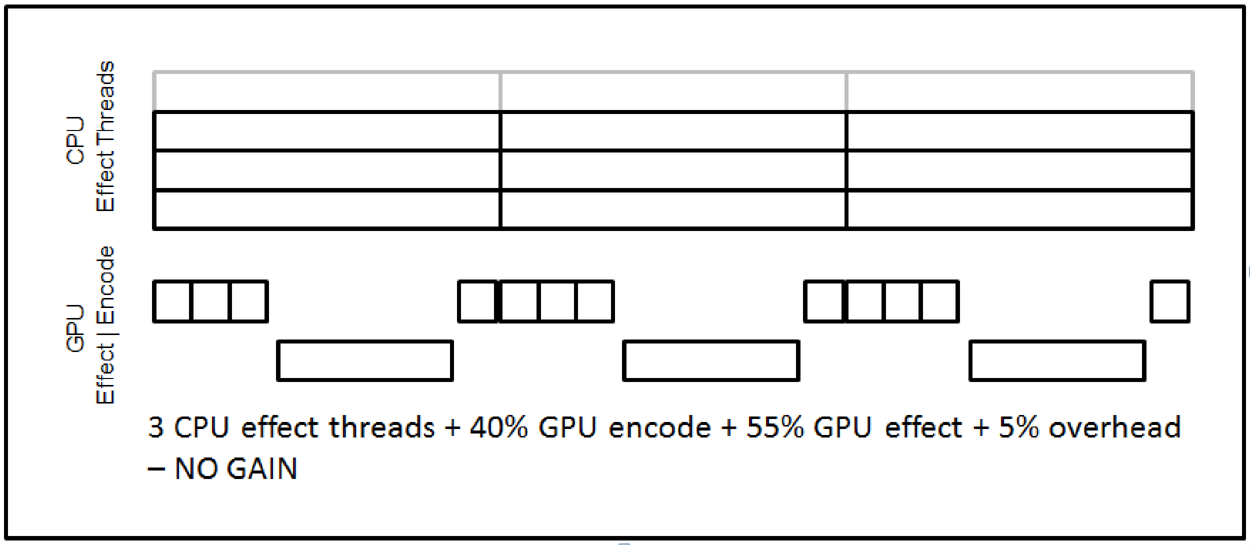
CPU Effect Threads
GPU Effect | Encode
3 CPU effect threads + 40% GPU encode + 55% GPU effect + 5% overhead – NO GAIN

Figure 4 – CPU and GPU effect processing and video encode over time

Furthermore, the fraction of execution unit time taken up by encode varies. If video encode on the execution units was the limiting factor, as might happen for very fast CPU effect algorithms, then encode might occupy 100% of the GPU execution unit time. With four threads processing video effects in parallel, the average time to produce one frame to be encoded is one fourth of the CPU effect time. So for the GPGPU video encode to not be an absolute limiting factor, the CPU effect time must be greater than 4x longer than the time required to encode a single video frame.

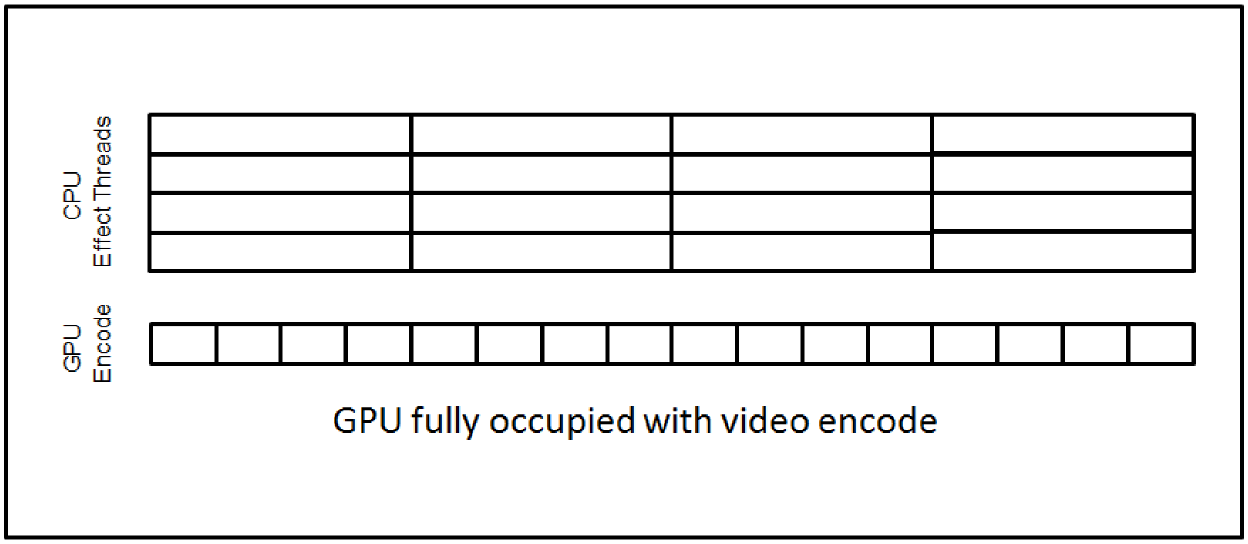

Figure 5

In the case described in “Isolating the Issue”, above, the GPGPU encode time averaged about 5 ms, though that would obviously vary depending on encode parameters and other factors. So CPU effect processing that takes less than 20 ms (4 times 5 ms encode time) would be video encode limited, with no chance of showing performance benefits from moving effect processing to the GPGPU. In the application in question, nearly half of the effects were simple enough to fall into that category.

But that’s still not the end of the story. We want to see performance gains by using the GPGPU for effects, e.g., a 25% performance gain. A 25% performance gain would mean 5 frames must be encoded in the same period of time. So the CPU effect would need to be at least 5 times longer than the GPGPU encode time (25 ms if the encode time is 5 ms).

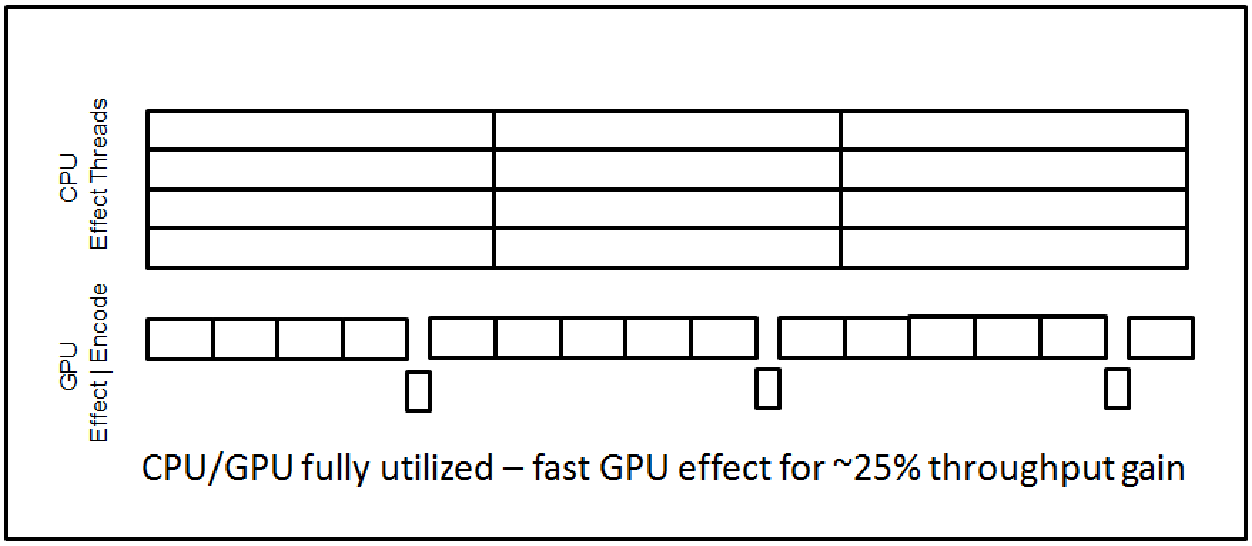
CPU
Effect Threads
GPU
Effect | Encode
CPU/GPU fully utilized – fast GPU effect for ~25% throughput gain

Figure 6

In addition there must be some time left over to execute the OpenCL GPGPU effect kernel processing. This gets a bit complex, so let's look at some examples and calculate the expected performance impact.

Examples of GPGPU Effect Processing Offload from CPU

Suppose again that the GPGPU video encode time is 5 ms, and that the CPU effect processing time is 30 ms. With 25 ms taken up by encode of 5 frames (to achieve the 25% performance gain described above), 5 ms are left for GPGPU effect processing. Since we're assuming one CPU thread is given up, the CPU will only produce 3 frames in 30 ms, leaving 2 frames for the GPU. Assuming 5% overhead, the GPGPU OpenCL kernel would need to process a frame in about 2 ms, i.e., 15 times faster than the same algorithm on a single CPU thread! This is not impossible, but it is fairly rare to achieve such an increase.

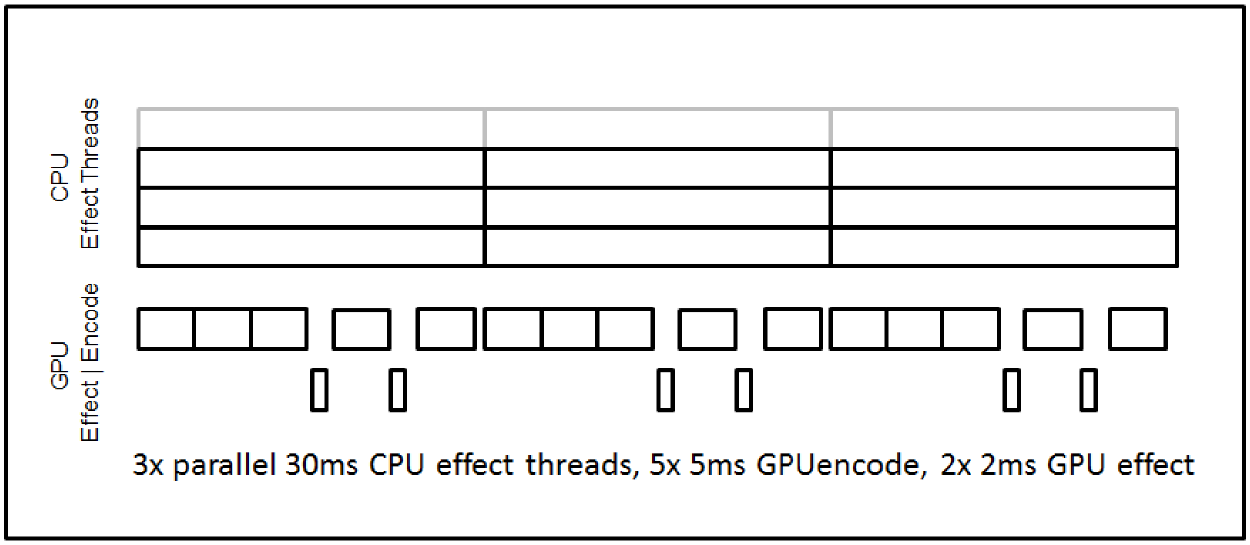
CPU
Effect Threads
GPU
Effect | Encode
3x parallel 30ms CPU effect threads, 5x 5ms GPUencode, 2x 2ms GPU effect

Figure 7

In the application in question, the average GPGPU speedup versus one thread of CPU was about 3x. Let's start with a more reasonable GPGPU effect processing time of 20 ms, with a CPU effect taking 60 ms. In 60 ms, with 3 frames to be encoded from the CPU and "X" from the GPU, and about 5% overhead of lost GPGPU execution time, the following equation must be solved for X to determine how many effect frames can be processed and encoded on the GPGPU in 60 ms:

$$60 \text{ ms} = (5 \text{ ms/frame encode})*(X+3 \text{ frames}) + (20 \text{ ms/frame effect})*(X \text{ GPU frames}) + 60*0.05$$

Equation 1

The solution for that is X = 1.2 frames of effect processing on GPGPU. This means that instead of 4 frames in 60 ms, with GPGPU effect processing, we can produce (1.2 + 3) = 4.2 frames—a 1.05x (5%) performance gain from a 3x kernel speed up.

What would it take to get a 25% performance gain? In the above equation, that would require X = 2, i.e., one frame to replace one CPU thread, plus another to get to five frames versus the original four in the same amount of time. The total encode time for 5 frames would be 25 ms (5 frames times 5 ms per frame). If we keep the same 60 ms CPU effect time, and let Y be the GPGPU effect time (and again with 5% or 3 ms of overhead time), the new equation looks like this:

$$60 \text{ ms} = (5 \text{ ms/frame encode})*(5 \text{ frame}) + (Y \text{ ms/frame})*(2 \text{ effect frames}) + 3 \text{ ms}$$

Equation 2

The solution for the GPGPU effect processing time is Y = 16 ms/frame. So to get a 25% application level speedup, the GPGPU OpenCL kernel will need to be 3.75x faster than the CPU. That’s not unreasonable to achieve.

Would it be possible to get a 2x application level speedup? Keeping the 60 ms CPU effect processing time for four frames, this would change the second equation above to require 8 frames output, 5 from the GPGPU processing and 3 from the CPU:

$$\boxed{60 \text{ ms} = (5 \text{ ms/frame encode})*(8 \text{ frame}) + (Y \text{ ms/frame})*(5 \text{ effect frames}) + 3 \text{ ms}}$$

Equation 3

This gives a GPGPU effect processing time of Y = 3.4 ms/frame, or a speed up of about 17.6x. Again, this degree of GPGPU gain over CPU performance is fairly rare.

Suppose the CPU effect processing time were 100 ms. This changes the equation (with 5% = 5 ms overhead) to:

$$\boxed{100 \text{ ms} = (5 \text{ ms/frame encode})*(8 \text{ frame}) + (Y \text{ ms/frame})*(5 \text{ effect frames}) + 5 \text{ ms}}$$

Equation 4

That yields a GPGPU effect processing time of Y = 11 ms/frame, or still requiring a 9.1x speed up to achieve a 2x performance gain. That is possible, but still pretty difficult to achieve.

As can be seen, the more the CPU effect processing time exceeds the GPGPU video encode processing time, the easier it is to achieve good performance gains by moving processing to the GPGPU.

Keeping the CPU Busy

What if the application had not eliminated a CPU thread when adding a GPU effect processing thread? This should be possible, as very little CPU time is required to run OpenCL kernels on the GPU, and to wait for the kernels to complete. Running effect processing on all available CPU cores will likely result in a bit poorer responsiveness in the OpenCL GPGPU threads. The CPU effect thread might need to finish its time slice before the GPU thread could respond to completion of a kernel and start the next kernel, resulting in poorer utilization of the GPU execution units.

Re-visiting the case of Equation 1, where the CPU takes 60 ms/frame and the GPU takes 20 ms/frame, Equation 5 below (with X as the number of frames produced on the GPU) covers the case of keeping 4 CPU threads processing effects, with 10% (6 ms) GPU overhead added to account for the delay in servicing the OpenCL kernels:

$$\boxed{60 \text{ ms} = (5 \text{ ms/frame encode})*(X+4 \text{ frames}) + (20 \text{ ms/frame effect})*(X \text{ GPU frames}) + 6 \text{ ms}}$$

Equation 5

Solving for the number of GPGPU effect frames, X = 0.97, or about one frame. So instead of the original 4.2 frames processed in 60 ms, the output would be about 5 frames, a 25% gain over CPU-only and a 1.19x improvement over the thread trade-off approach of Equation 1. But note that this gain still relies on a respectable 3x GPGPU speedup over the CPU, and a CPU effect time that is fairly large compared to the GPGPU video frame encode time. If the CPU time had been only 30 ms, the equation would have been:

$$\boxed{30 \text{ ms} = (5 \text{ ms/frame encode})*(X+4 \text{ frames}) + (20 \text{ ms/frame effect})*(X \text{ GPU frames}) + 3 \text{ ms}}$$

Equation 6

That would yield GPGPU effect frames of about X = 0.25, for 4.25 frames total, or only about a 6% gain over CPU-only. So even with the CPU continuing to process effects, the CPU effect processing time must be quite a bit higher than the GPGPU video encode time, to have a chance of good performance gains.

Intel Turbo Mode

At this point, it is important to mention an interesting factor—the idea of turbo mode. Turbo mode essentially allows either the CPU or the GPU to run faster, when the other is not so busy. The GPU frequency can almost double if the CPU isn't very busy, while the CPU might increase its frequency by a lesser amount. Both cannot increase their frequencies, as this capability is limited by thermal properties of the processor—both running at increased frequencies would make the chip run too hot.

When running in turbo mode, the simple calculations above can't be taken literally, especially if the CPU is left fully loaded with active threads while trying to gain performance from the GPGPU. In general, you can expect that this will mean that the straight-forward calculations above somewhat over-estimate the potential performance to be gained by adding GPGPU OpenCL processing to an application, if the calculations are based on separate CPU and GPU performance timings.

Conclusions

Application performance tuning using both GPGPU accelerated OpenCL and Intel Quick Sync Video must be done with the awareness that those GPGPU processes will compete for GPU processing resources. This has several key implications:

- For a 3rd generation Intel Core processor with four cores running threads that feed frames to an Intel HD Graphics 4000 integrated GPU for video encode, no performance gains are possible by off-loading CPU processing to a GPGPU OpenCL kernel if the CPU processing time is less than four times the GPU video encode time for a frame. An analogous rule will apply for other core counts. (Use Intel GPA to measure the video encode time on the programmable graphics execution units.)

- The CPU processing time (on a fully utilized four core CPU) must be more than four times longer than the GPGPU video encode processing time, per frame, in order to leave enough GPGPU time for work off-loaded to a GPGPU OpenCL kernel. It generally must be substantially longer than the equivalent GPGPU processing time, in order to attain reasonably good (25% or better) performance gains. The longer the CPU processing time, the easier it is for OpenCL on GPGPU to provide good performance benefits.

- Intel Quick Sync Video encode may need to do some processing on the GPU. This will reduce the amount of processing available for OpenCL processing on the GPU, which in turn can make it more difficult to achieve performance gains by off-loading processing from the CPU

- For best performance, keep the CPU fully loaded when adding GPGPU OpenCL processing. (But if lower power consumption is desired over performance, consider moving processing off the CPU. GPGPU processing can be substantially more power efficient.)

- Performance will not increase exactly according to the simple algebraic equations used above, due to Turbo mode frequency scaling. The equations should be considered as an upper-limit approximation of performance gains, or a lower limit on the speed-up factor required for a GPGPU OpenCL kernel to provide a specified application level performance gain.

Observing these factors could save a developer significant time when performance tuning applications that make use of Intel Quick Sync Video encode, or, for that matter, any dual use of Intel programmable graphics execution units. By measuring the time spent in CPU threads, and using the Intel GPA Media Performance Analyzer to measure the time of existing (Intel Quick Sync Video or other) GPGPU processing, the developer can quickly estimate whether moving CPU processing to Intel GPGPU using OpenCL will provide useful performance benefits.

Additional Resources

Intel® SDK for OpenCL* Applications website: www.intel.com/software/opencl

Intel® SDK for OpenCL* Applications 2012 - Optimization Guide:
http://software.intel.com/sites/landingpage/opencl/optimization-guide/index.htm

Intel Graphics Performance Analyzers: www.intel.com/software/gpa

Acknowledgements

Thanks to Brijender Bharti, Eric Sardella, Jeffrey Mcallister, and Chuck DeSylva for review and suggestions.

About the Author

Tom Craver is an Intel application engineer in Chandler, Arizona. He is currently focused on performance of applications of OpenCL on Intel processor graphics, but has extensive background in SIMD and threaded parallel coding for performance, mostly applied to media applications such as video and audio codecs and video effect processing.

Notices

Optimization Notice

NOTE: additional notices and disclaimers may be needed, depending on the content of the collateral. Generally, they can be found in the following areas, and must be appended to the Notices section of the paper.

General Notices: *Notices placed in all materials distributed or released by Intel*

Benchmarking and Performance Disclaimers: Disclaimers for Intel materials that use benchmarks or make performance claims.

Technical Collateral Disclaimers: *Disclaimers that should be included in Intel technical materials that describe the form, fit or function of Intel products.*

Technology Notices: *Notices for Intel materials when the benefits or features of a technology or program are described. Note for technology disclaimers - if every product being discussed (e.g., ACER ULV) has the particular technology/feature, then you can remove the requirements statement in the disclaimer. If you have multiple technical disclaimers, you can consolidate the "your performance may vary" statements and only put in a single "your mileage may vary".*