CHAPTER 8

Performance, Power, and Quality Tradeoff Analysis

When you're operating under resource constraints, it is necessary to understand the right tradeoffs to reach your goal. Often one thing must be given up to gain another thing. Depending on the objectives, priorities, and tolerances of a solution, an appropriate balance must be struck between the best use of available resources and the best achievable success measure.

In the case of creating compressed video, measures are used to obtain the best quality and the highest performance at the cost of the least number of bits and the lowest power consumption. For an overall encoding solution, success criteria may also include choices among hardware-based, software-based, and hybrid encoding systems that offer tradeoffs in flexibility, scalability, programmability, ease of use, and price. Users of video coding solutions, therefore, need to be aware of the appropriate choices to be made among cost, adaptability, scalability, coding efficiency, performance, power, and quality so as to achieve a particular video coding solution.

Tradeoff analysis is useful in many real-life situations. Understanding the options, particularly in terms of performance, power, and quality, is a valuable capability for architects, developers, validators, and technical marketers, as much as it is helpful for technical reviewers, procurers, and end-users of encoding solutions. Making informed product decisions by assessing the strengths and weaknesses of an encoder, comparing two encoders in terms of their practical metrics, and tuning the encoding parameters to achieve optimized encoders are among the decision points offered by such analysis. Furthermore, when new features are added to an existing encoder, such analysis can reveal the costs and benefits of those new features in particular measures. This helps users decide whether or not to enable some optional encoding features under various constraints and application requirements.

As coding efficiencies in terms of rate distortion of various algorithms were covered in previous chapters, here we turn the discussion toward an examination of how tradeoff analysis actually works. We focus on three major areas of optimization and the tradeoffs inherent in them—namely, performance, power, and quality. These three areas are of critical importance in present-day video encoding usage models and encoding solutions.

The discussion starts with the common considerations of tradeoff analysis involving these three measures, along with other options that may appear. This is followed by a discussion of the effects of encoding parameter tuning on these three measures. We then briefly discuss a few common optimization strategies and approaches.

With these discussions we present case studies of tradeoff analysis that look at performance power, performance quality, and power quality. These examples view the variables from several different points of view, shedding light on the methodologies commonly used in such analyses.

Considerations in a Tradeoff Analysis

Tradeoff analyses are essentially decision-making exercises. With the wide variability of video complexities and the numerous combinations of tuning parameters available, tradeoffs are commonly made based on the application's requirements for enhanced visual experience. The outcome of a tradeoff—namely, the properties of the encoded bitstream—determine the worthiness of the analysis. It is imperative that an encoded bitstream be syntactically valid, but its merit is typically judged in terms of its properties, including the amount of compression, the perceived quality when decoded, the amount of time it took to generate it, and its power consumption.

An important application of tradeoff analysis is a comparison of two encoding solutions based on both's performance, power, and quality. To make a fair comparison in such cases, various system parameters must be considered and made as equivalent as as possible. Such considerations include:

- The configurable TDP (cTDP) or scenario design power (SDP) settings (in fourth-generation Intel Core or later processors) such as nominal TDP, cTDP up, or cTDP down, which are usually done to accommodate overclocking or available cooling capacities.
- Power mode, whether AC (plugged in) or DC (battery).
- Operating system graphics power settings and power plans, such as maximum battery life and balanced or maximum performance.
- Display panel interface, such as embedded display port (eDP) or high-definition multimedia interface (HDMI).
- Display resolution, refresh rate, rotation and scaling options, color depth, and color enhancement settings.
- Number and type of display units connected to the system (e.g., single or multiple, primary or secondary, local or remote).
- Overhead and optimizations, application's settings for color correction and color enhancement, driver settings, and so on.
- Firmware, middleware, and driver versions.
- Operating system builds and versions.

- Operating voltage and frequency of the CPU and GPU.
- Memory configuration and memory speed.
- Source video content format and characteristics.

When conducting these comparisons, it is also necessary to turn off irrelevant applications or processes, leaving the test workload as the only one running on the platform. This ensures that the available resources are properly allocated to the workload and there is no resource contention or scheduling conflict. Futhermore, it is generally good practice to take the same measurement several times and to use the median result. This reduces any noise that may be present within the measurement tolerance. Keeping the system temperature stable is also necessary to reduce potential noise in the measured data; temperature contollers that automatically activate a dedicated fan when a higher than target temperature is sensed are typically used for this purpose.

Types of Tradeoff Analyses

For mobile devices, saving power is typically the high priority. Therefore, if a decision only moderately impacts visual quality but saves substantial power, that quality tradeoff in favor of power saving is usually preferred for low-power mobile devices. Generally, techniques that provide greater compression while keeping the visual quality level approximately the same are good candidates for tradeoffs. However, these techniques often come with higher complexity, imposing a greater demand on power or performance. For example, HEVC encoding offers improved efficiency at the cost of more complex compression compared to AVC. As such, HEVC encoding in lieu of AVC encoding for an HD video is not an automatic choice on a power-constrained mobile device. Thus, tradeoff analysis must consider the overall benefit or the net gain.

Priorities are driven primarily by the requirements of the usage models and hence they also govern the tradeoffs that are made. For example, consider a videoconferencing application versus a video transcoding application. For the former, the low delay and real-time requirements demand steady power consumption throughout the video session, while for the latter, a run-fast-and-sleep approach is more beneficial. Additional limits may be applied depending on the availablility of resources. For instance, certain techniques that trade visual quality for better performance may not always be feasible, owing to limitations of the system, including the TDP, maximum processor frequency limit, and so on. Similarly, although a low-level cache can increase performance in many video applcations, it may not be available in the system under consideration.

Effects of Parameter Tuning

Various encoding parameters affect the relationship between performance and power consumption, as well as visual quality. The motivation for such tuning parameters is often to expose opportunities for obtaining higher performance, better quality, or power savings. Further, these tuning exercises reveal whether there are inefficiencies in a non-optimized video application, in addition to potential causes for such inefficiencies, all of which can lead to better solutions.

Typically, the impact of such tuning is more easily seen in improved visual quality and performance, rather than in lowered power consumption. As elaborated in Chapters 4 and 5, many parameters have a significant impact on both performance and quality, including the video spatial resolution, frame rate, and bit rate; group of pictures structure; number of reference pictures; R-D optimization in mode decision and determination of motion vectors; adaptive deblocking filter; various levels of independent data units such as macroblocks, slices, frames, or group of pictures; multiple passes of analysis and processing, multiple generations of compression, and pre- and post-processing filters; and special-effects filters.

Some of these parameters have greater effects on the visual quality while others benefit performance and power; your parameter tuning efforts should take these relative benefits into account. For example, using B-pictures significantly affects both visual quality and performance, but using R-D optimization in mode decision and determination of motion vectors slows down the encoding speed more significantly than it improves visual quality. Similarly, using multiple slices slightly reduces visual quality, but it improves parallelizability and scalability and it offers performance and power-saving opportunities.

In addition, it is important to consider just how *much* power savings or improved performance can be achieved while the encoded video retains reasonable visual quality. The nature of the video content and the bit allocation policy are important considerations here.

Optimization Strategies

There are a few optimization strategies with regard to improving performance or saving power, usually without surrendering visual quality. These strategies are typically employed in optimizations of video coding applications and come with appropriate tradeoffs.

- **Reducing scheduling delay:** Batching allows a series of function calls, such as motion estimation calls, for various macroblocks to be done together. Further, it allows macroblock-row-level multithreading for appropriate parallel processing. Typically, all operations within a batch share the same memory surface, thereby improving data fetch and cache hit rate. However, the application must wait for the writes to complete for all macroblocks in a batch before it can read from the memory surface. This introduces a small delay, but that delay is nonetheless suitable for video applications such as video streaming. The performance benefits achieved by batching do not typically sacrifice visual quality, and they give plenty of headroom for other workloads running concurrently.

- **Optimizing slack time:** Proactive energy optimization by workload shaping[1] is another way to obtain power optimization. As opposed to worst-case design philosophy, the video codec implementation framework not only is aware of hardware-specific details but also proactively adapts the implementation strategy to offer the best possible resource utilization. When it's in a traditional reactive energy optimization approach, the system merely adjusts its execution speed to the changing workload by exploiting available slack time. In the proactive scheme, the implementation can alter the shape of the workload at a given time, thereby achieving ~50 to 90 percent more energy savings than traditional implementations. In this case, the slack is accumulated over multiple data units so that the underlying processor can use a more aggressive power-saving approach, such as a deep sleep, for the larger slack period. Further, by reordering the video frames within a tolerable latency increase, additional slack accumulation and consequent power savings are achieved. The proactive workload adaptation is done by using high-level complexity models, while the video codec framework interprets the models at run-time to choose the appropriate frequency and voltage of operation, thereby minimizing energy without loss in quality and without missing any deadlines for a frame.

- **Parallelizing tasks:** The benefits of the parallelization of tasks, data, and instructions have been discussed in Chapter 5. Parallelizing independent tasks and distributing them over multiple processors makes full use of available processing capabilities. This allows the processing to complete quickly and enables the processors to go to deeper sleep states for longer periods of time, thus achieving power savings. Pipelines of tasks also keep the resources busy for as long as necessary and minimize resource conflicts. Further, with appropriate design of parallel applications, bottlenecks can be removed by re-scheduling a task to a different processor when a processor becomes too slow or unresponsive. Prioritization of tasks on various processors also helps overall performance. However, parallelization has its potential disadvantages of added overhead, such as inter-processor communication or synchronization costs.

[1]V. Akella, M. van der Shaar, and W. F. Kao, "Proactive Energy Optimization Algorithms for Wavelet-based Video Codecs on Power-aware Processors," in *Proceedings of IEEE International Conference on Multimedia and Expo* (Amsterdam, The Netherlands: IEEE, July 2005), 566–69.

- **Optimizing I/O:** Besides maximizing the use of system resources for the shortest possible time, increasing the data access speed and reducing I/O bottlenecks have prime significance when making power versus processing delay choices. In some off-line video applications, such as cloud-based video distribution, it is possible to obtain better overall power profiles by using dedicated I/O processors while groups of parallel processors are encoding batches of video segments. However, a side effect of this technique is the increased delay; the final video bitstream can only be stitched together when all the encoding processors are done. Therefore, the number of groups served by the I/O processor becomes the parameter of a possible tradeoff. Data prefetching and streaming opportunities should also be exploited as much as possible, noting that video data is particularly amenable to such techniques.

- **Reducing compute operations:** Algorithmic optimization allows techniques such as threshold-based early termination of loops, or exploitation of SIMD-style parallelism. These techniques help reduce the number of compute operations. However, this requires extremely careful analysis to determine and understand the various tradeoffs involved; in some cases, the visual quality may be affected as well. Furthermore, optimizing the code by hand or using the various compiler optimization techniques has direct impact on performance and power consumption by reducing the number of instructions to be executed.

- **Optimizing the cost vs. benefit:** The cost of high performance in terms of implementation complexity, and consequently in terms of power consumption, should be always carefully considered. It may be necessary to redesign the performance optimization approaches to tackle power consumption. In an image-filtering experiment,[2] it was observed that the behavior of this workload is drastically different from, for example, a scalar-vector multiplication-accumulation workload, although both workloads are similarly parallelizable. In the image-filtering case, performance optimizations are possible owing to the available data-reuse opportunities. This is also true for many video coding and processing applications. However, the energy profile of the image-filtering process is irregular, owing to uneven data reuse and resource scheduling. The energy optimal point corresponds to a large unrolling factor, relatively modest array partitioning,

[2]B. Reagen, Y. S. Shao, G. Y. Wei, and D. Brooks, "Quantifying Acceleration: Power/Performance Trade-Offs of Application Kernels in Hardware," in *Proceedings of 2013 IEEE International Symposium on Low Power Electronics and Design* (Beijing: IEEE, September 2013), 395–400.

pipelined multipliers, and non-pipelined loops. The large unrolling factor allows for greatest utilization of loaded data for the reasonable bandwidth requirements. The bandwidth needs are usually amortized over multiple cycles, maximizing the reuse and efficiency of given resources. This results in a complex control flow and a non-intuitive energy optimal design solution. For such complex workloads, the cost of additional power consumption by a higher performing design may not always be justified.

The Performance–Power Tradeoff

Recall from Equation 6-1 that power is a linear function of frequency. As improved performance directly depends on increased frequency up to a certain frequency limit, it is desirable to increase the frequency while keeping the power consumption the same. To achieve this, the co-factors of frequency—namely, the voltage, the capacitance, and the activity factor—need to be reduced. Furthermore, leakage current needs to be reduced. The activity factor is typically reduced by using clock gating, while the capacitance is reduced by downsizing the gates. As mentioned in Chapter 6, lowering the voltage can only be done in the voltage scaling region until a minimum voltage is reached, which must be sufficient for transistors to operate. Note from Figure 6-7 that leakage is constant in the low-frequency V_{min} region, while in the voltage-scaling region the leakage keeps increasing at typical operating points. Leakage current is decreased by reducing the transistor width and using lower leakage transistors. However, lower leakage transistors are also slower compared to leaky ones. Therefore, hardware designers need to make the appropriate optimizations to maximize the frequency for a given amount of power consumption.

It is important to note that ~90 percent of all modern mobile platforms are power limited. Therefore, every bit of power savings is considered equivalent to a corresponding gain in frequency. Typically, in the V_{min} region of low-power platforms, ~10 percent power saving translates to some 15 percent gain in frequency, while in the voltage scaling region, ~20 percent power saving corresponds to a mere ~5 percent gain in frequency. Thus the importance of judicious hardware design for optimal voltage, capacitance, activity factor, and leakage cannot be overstated, particularly with regard to obtaining the highest frequency at a given power budget. Operating at higher frequency generally implies better performance for various applications.

For video applications, higher CPU or GPU operating frequencies provide faster encoding speed, but they also consume more energy. A tradeoff between energy consumed and encoding speed is thus necessary at the system-design and hardware-architectural level, particularly for GPU-accelerated encoders. The programmable part of the encoding should also maintain an appropriate balance between performance and power. Usually this is done by parallelizing the encoding tasks, by scheduling appropriate tasks among multiple threads of CPU and GPU, by migrating tasks between the CPU and the GPU on the fly, by adjusting the schedules of the tasks, and/or by optimizing resource utilization for individual tasks, all without significantly affecting visual quality. For example, depending on the complexity of the video content, encoding two slices of a picture in parallel can yield ~10 percent performance gain with negligible quality impact. Tuning of encoding parameters also affects the overall encoding speed and power consumption, as some hardware units, such as the

bit-rate control units, scaling units, and so on, may be optionally turned off depending on the parameter setting. Such tuning, however, may influence visual quality.

Let's consider the following case study of a performance–power tradeoff for a video transcoding application. For comparison, the same tests are run on two platforms with different performance-power characteristics. Note that the transcoding comprises decoding of a compressed video into an uncompressed format, which is subsequently encoded using appropriate encoding parameters into the target video in compressed format. The decoding tasks in the transcode operation remain the same for the same source video content, and usually the decoding is much faster than the encoding. Thus, the overall transcode performance can be measured in terms of the encoding speed alone. As such, the terms *performance* and *encoding speed* are used interchangeably.

Case Study

Consider two transcoding workloads, each about 5 minutes long; both of them operate with a spatial resolution of 1920×1080p at 30 fps, and transcode from higher bit-rate H.264 input bitstreams into lower bit-rate bitstreams of the same format. Workload 1 consists of lower complexity frames with infrequent scene changes, fewer details, and slower motion compared to Workload 2.

The transcoding tasks are carried out on two Intel platforms. Table 8-1 shows the platform configurations.

Table 8-1. *Plaform Configuration for Transcode Experiment*

	Platform 1	Platform 2
Processor	4th-gen. Core i7	4th-gen. Core i5
# Cores	4	2
CPU frequency	2 GHz	1.3 GHz
CPU turbo	3.2 GHz	2.6 GHz
TDP	47 W	15 W
Cache size	6 MB	3 MB
Graphics	Intel (R) Iris Pro (TM) 5200	Intel (R) Iris Pro (TM) 5000
Max GPU frequency	1.2 GHz	1 GHz
Embedded DRAM	Yes	No
Memory	4 GB dual channel	4 GB dual channel
Memory speed	1333 MHz	1333 MHz

We also consider three sets of encoding parameters, numbered 1, 2, and 3. In each set, a combination of encoding parameters is used. Table 8-2 shows some of the important distinctions for each parameter set.

Table 8-2. *Important Differences in Settings for Each Parameter Set*

	Param Set 1	Param Set 2	Param Set 3
Configuration	Fast	Faster	Fastest
Motion estimation algorithm	Algorithm 1	Algorithm 1	Algorithm 2
Motion estimation search range	48 pixels	48 pixels	28 pixels
Fractional pixel motion compensation	1/8 pixel	1/8 pixel	1/4 pixel
Adaptive search	Yes	No	No
Multiple reference pictures	Yes	No	No
Multiple predictions	Yes	No	No
Macroblock mode decision	Complex	Complex	Simplified

Figure 8-1 shows transcode performance on the two platforms using the two workloads with various sets of encoding parameters. It is notable that, owing to the different complexities of the two workloads, the tuning of parameters affects them differently. It is also observed that the degree of such impact is different on the two different platforms.

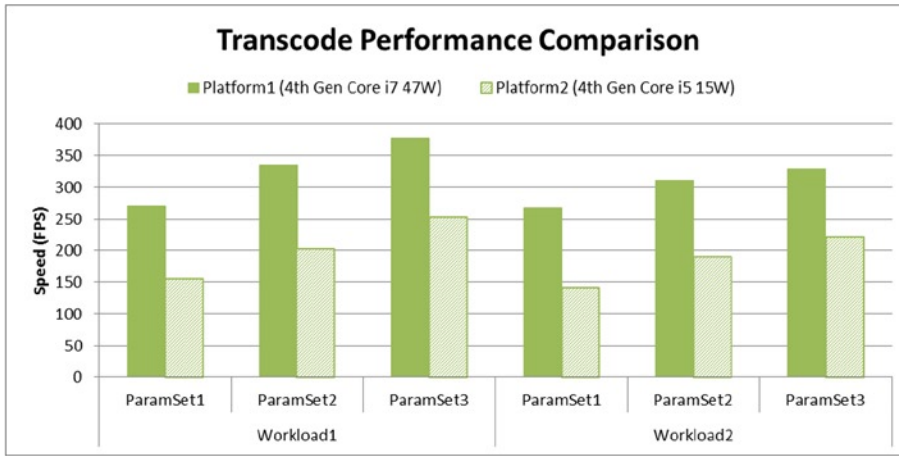

Figure 8-1. *Transcode performance comparison of two platforms*

From Figure 8-1 it can be noted that Platform 1 has an average of ~60 percent better throughput in terms of encoding speed compared to Platform 2, of which the embedded dynamic RAM provides ~10 percent performance throughput difference and the GPU frequency difference accounts for another ~20 percent. The remaining ~30 percent difference can be attributed to a combination of processor graphics hardware optimization, number of GPU execution units, cache size, number of CPU cores, CPU clock speed, turbo capacity, and so on.

While Workload 2 gives consistently increasing performance as the parameters move from fast to fastest cominations, especially on Platform 1, Workload 1 provides a peak performance of over 12-fold faster than real-time speed with parameter set 3. Therefore, it is clear that workload characteristics, along with parameter tuning, greatly influence the transcode performance. Comparing the fastest parameter set (3) for both workloads on Platform 1, it can be observed that Workload 1 provides ~13 percent better performance compared to Workload 2. On Platform 2, a similar trend is observed, where Workload 1 is ~12 percent faster compared to Workload 2.

Note that, owing to the characteristics of Workload 2 and to the constrained resources on Platform 2, parameter set 1 yields significantly lower performance on this platform because this parameter set includes multiple reference pictures, multiple predictions, and elaborate analysis for mode decisions.

Figure 8-2 shows the package power consumptions by the two platforms for the two workloads with the same sets of parameters.

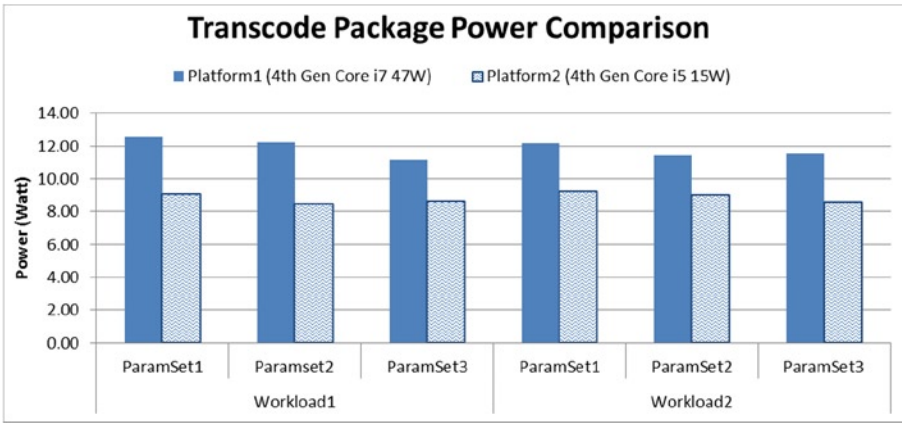

Figure 8-2. *Transcode package power consumption comparison of two platforms*

From Figure 8-2, it is clear that on, average, Platform 1 consumes ~34 percent more package power compared to Platform 2, while neither platform reaches its maximum TDP limit for the workloads under consideration. However, some parameter settings require certain hardware units to turn on and consume power, while others don't. This is evident from the difference in power consumption between the two platforms, ranging from ~14 percent to ~44 percent.

Interesting observations can also be made if the absolute power consumption is considered on each platform. As the parameters are tuned, the power consumption generally decreases, especially on Platform 1. Further, on Platform 1, Workload 1 has a 28 percent dynamic range of power consumption, while Workload 2 has a mere 8 percent dynamic range. On Platform 2, however, these numbers are ~7 and ~10 percent, respectively. This shows that on Platform 1, the power consumption of Workload 1 reacts more quickly to changing parameters compared to Workload 2. However, on Platform 2,

these workloads are not compute-bound and therefore do not react to changing parameters. In this case, cache performance and number of GPU execution units become the dominant factors, with little regard to the encoding parameters.

Figure 8-3 shows the platform efficiency in terms of fps per watt for both workloads on the two platforms for each set of parameters. Platform 1 is generally more efficient than Platform 2, with an average of ~23 percent better efficiency, owing to its dedicated embedded dynamic RAM, higher GPU frequency, higher number of GPU execution units, and better cache performance.

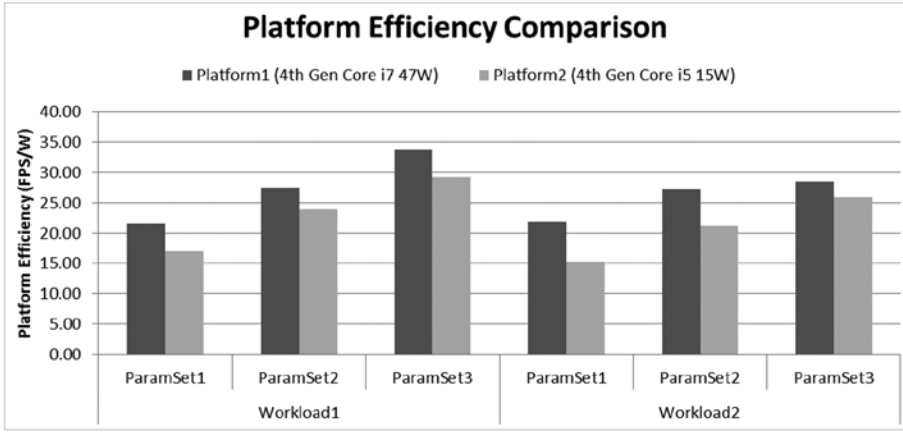

Figure 8-3. *Platform efficiency in terms of fps per watt during a set of transcode experiments*

From Figure 8-3 it is also observed that parameter tuning somewhat similarly impacts Workload 1 on both platforms, but for Workload 2, Platform 2 shows larger variation in terms of platform efficiency. This behavior of platform efficiency is not only due to changing parameters but also to the different characteristics of the workloads.

Figure 8-4 shows another point of view for performance versus power analysis. The two platforms are clearly showing different performance characteristics owing to differences in their available resources. Both workloads are clustered together on the two platforms. Because of the bigger cache size and the presence of an embedded dynamic RAM, Platform 1 generally consumes more power compared to Platform 2, but it provides much higher performance as well.

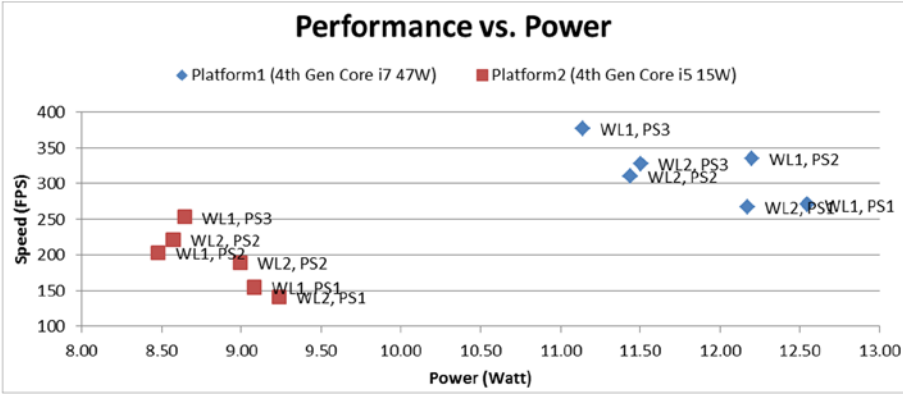

Figure 8-4. *Performance vs. power on the two platforms (WL and PS are abbreviations of workload and parameter set, respectively)*

From Figure 8-4 it can be observed that, on a given platform, appropriate parameter selections can provide good power-saving opportunities. For example, on Platform 2, Workload 1 can provide close to 1 watt of power saving using parameter set 3 compared to parameter set 1.

Note that, while one is performing a performance and power tradeoff analysis, employing parallelization techniques or optimizing resource utilization generally has little impact on visual quality. However, by tuning the encoding parameters, the quality is affected as well. In these cases, the power–performance tradeoff becomes a power–performance–quality three-way tradeoff. If the bit-rate control algorithm tries to maintain the same quality with a variable bit rate, resulting in different bitstream sizes, then the tradeoff becomes a power–performance–encoding efficiency three-way tradeoff. This is a side effect of the power–performance tradeoff.

The Performance–Quality Tradeoff

Higher encoding speed can be obtained by manipulating some video encoding parameters such as the bit rate or quantization parameter. By discarding a large percentage of high-frequency details, there remains less information to be processed, and thus encoding becomes faster. However, this directly affects the visual quality of the resulting video. On the other hand, using B-pictures offers a different performance-quality factor. Although a delay is introduced as the reference frames must be available before a B-picture can be decoded, the use of B-pictures generally improves the visual quality as well as the temporal video smoothness. For example, in a set of experiments with the H.264 encoding, we found that when we used two B-pictures between the reference pictures, the average impact on FPS was ~7 percent, but that some ~0.35 dB better quality in terms of BD-PSNR was obtainable for the same set of HD video sequences.

Similarly, manipulating parameters such as the motion search range, search method, number of reference pictures, two-pass encoding, and so on can impact both performance and quality. Therefore, it is necessary to always look into the potential impact on visual quality of a any performance gain or loss before considering a feature or parameter change in the video encoder. To illustrate the performance–quality tradeoff, we present two case studies and discuss the results obtained.

Case Study I

A 35 Mbps H.264 input bitstream is transcoded into another bitstream of the same format, but with a lower bit rate of 7 Mbps. The original video test clip is about 5 minutes long, with a spatial resolution of 1920×1080p at 30 fps. It comprises several scenes with varying complexities ranging from high spatial details to mostly flat regions, and from high irregular motion to static scenes. The transcoding tasks involve fully decoding the bitstream and re-encoding it with new coding parameters.

The transcoding tasks are carried out on a platform with configurations given in Table 8-3.

Table 8-3. *Platform Configuration for Transcoding in Case Study I*

System Parameter	Configuration
Processor	4th-gen. Core i5
# Cores	4
CPU frequency	2.9 GHz
CPU turbo	3.6 GHz
TDP	65 W
Cache size	6 MB
Graphics	Intel (R) HD Graphics (TM) 4600
Max GPU frequency	1.15 GHz
Embedded DRAM	Yes
Memory	4 GB dual channel
Memory speed	1333 MHz

Two transcoder implementations are used: a software-based transcoder running entirely on the CPU, and a GPU-accelerated transcoder where most of the compute-intensive tasks are done in special-purpose fixed-function hardware units. The two implementations optimize the parameters differently, but both offer three output modes of performance–quality tradeoffs: the best quality mode, the balanced mode, and the best speed mode.

Although the GPU-accelerated implementation provides only a few externally settable parameters, and while there are many choices available for the CPU-only implementation, effort is made to keep these paramteters as close as possible for both implementations. Surely, there are variations in the exact parameters that are tuned for a mode by the two implementations, but there are some commonalities as well. Table 8-4 summarizes the common parameters.

Table 8-4. *Common Parameters of the Two Implementations*

Parameter	Best Quality Mode	Balanced Mode	Best Speed Mode
Motion estimation and mode decision methods	Algorithm 1	Algorithm 2	Algorithm 3 (with early exits)
Fractional motion compensation	Eighth pixel	Quarter pixel	None
Reference pictures	Many	Few	Single
Adaptive search	Yes	No	No
Motion search range	Large	Medium	Small
Weighted prediction	Yes	Yes	No
Multiple B-pictures	Yes	Yes	No
Sub-macroblock partitions	All	Few	None
Scene change detection	Yes	Yes	No
Look-ahead analysis for bit rate control	Many frames	Few frames	No look-ahead

Note that these parameters are used slightly differently in the two implementations, so the exact same quality is not expected from the two implementations. Also note that the focus of the GPU-accelerated implementation is on achieving higher performance without losing much visual quality, thus only a few parameters are varied from the best quality to the best speed in this implementation. On the other hand, obtaining higher performance is difficult in CPU-only implementation; therefore, the best speed mode in this implementation turns off several features much more aggressively compared to the GPU-accelearted implementation.

The performance is measured in terms of FPS for the three modes of operation for both transcoder implementations. Note that the coding parameters are tuned for each of the three modes to obtain certain performance–quality tradeoffs. Figure 8-5 shows the transcode performance comparison between the CPU-only and the GPU-accelerated implementations. It also shows speedups of the different modes.

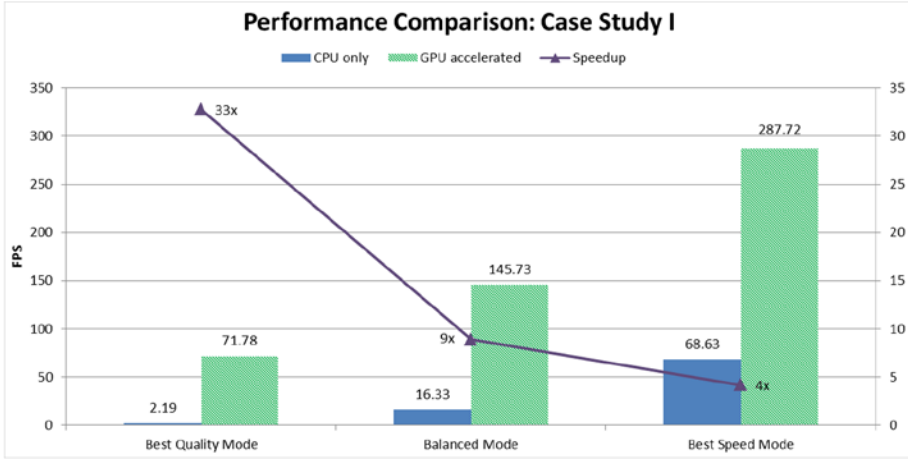

Figure 8-5. *Transcode comparison of various performance modes*

From Figure 8-5, we can see that both implementations scale in terms of speed from the best quality, to the balanced, to the best speed modes. For instance, the GPU-accelerated implementation speeds up the encoding from one mode to the next by a factor of approximately 2. However, with more aggressive tuning of the encoding parameters, the CPU-only implementation scales from the best quality to the balanced mode by performing the optimizations given in Table 8-5 and achieving a 7.45 times speedup. Similarly, from the balanced to the best speed mode, an additional 4.2 times speedup is obtained.

Table 8-5. *Optimizations in Different Modes for the CPU-only Implementation*

Parameters	Best Quality	Balanced	Best Speed
Motion estimation method	Uneven multihexagon search	Hexagonal search with radius 2	Diamond search with radius 1
Maximum motion vector range	24	16	16
Sub-pixel motion estimation	Yes	Yes	No
Partitions	All (p8x8, p4x4, b8x8, i8x8, i4x4)	p8x8, b8x8, i8x8, i4x4	No sub-macroblock partitions
Use trellis for mode decisions	Yes	No	No
Adaptive quantization	Yes, with auto-variance	Yes	No

(*continued*)

Table 8-5. (*continued*)

Parameters	Best Quality	Balanced	Best Speed
R-D mode decision	All picture types	I-picture and P-picture only	None
Max number of reference pictures	16	2	1
Number of references for weighted prediction for P-pictures	2	1	None
Number of frames to look-ahead	60	30	None
Max number of adaptive B-pictures	8	2	No B-pictures
CABAC	Yes	Yes	No
In-loop deblocking	Yes	Yes	No
8×8 DCT	Yes	Yes	No
Scene change detection	Yes	Yes	No

Obviously, these optimizations take a toll on the visual quality, as can be observed from Figure 8-6, which shows the quality comparisons for the two implementations. From the best quality to the best speed, the CPU-only implementation loses on an average of about 5 dB in terms of PSNR, with a tiny reduction of less than 0.1 percent in file size. On the other hand, with the focus on performance improvement while maintaining visual quality, the GPU-accelerated implementation does a good job of losing only an average of about 0.6 dB of PSNR from the best quality to the best speed mode. However, this implementation ends up with a ~1.25 percent larger file size with the best speed mode compared to the best quality mode, thereby trading off the amount of compression achieved.

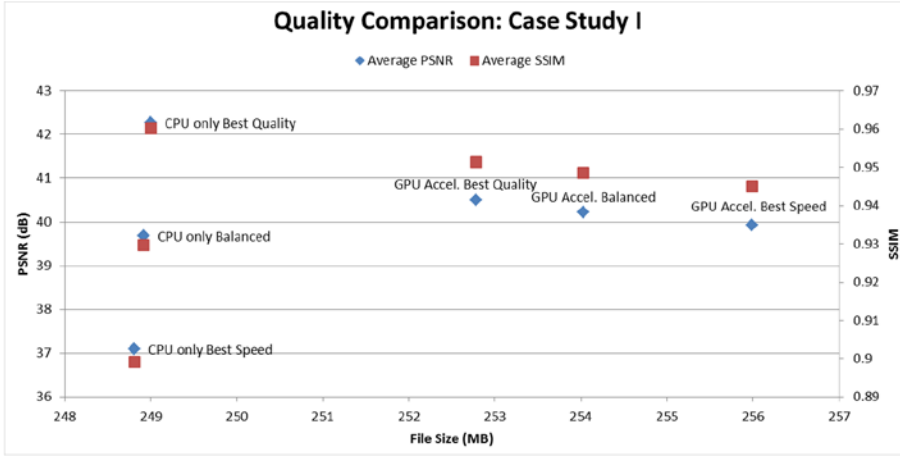

Figure 8-6. *Quality comparisons of the two implementations*

Another observation can be made from Figures 8-5 and 8-6; in terms of speed for the three performance modes, the GPU-accelerated implementation is faster than the CPU-only implementation by factors of approximately 33, 9, and 4, respectively. This shows the contrast between the two implementations in terms of parameters tuning. While the GPU-accelerated implementation starts with a much better performance in the best quality mode, it has an average of 1.76 dB lower PSNR with a ~1.5 percent larger file size compared to the best quality mode in the CPU-only implementation. Thus, it has already sacrificed significant visual quality in favor of performance. Further, the GPU-accelerated implementation is less flexible in terms of ability to change the algorithms, as some of the algorithms are implemented in the fixed-function hardware units. Nonetheless, in the best speed mode, this implementation shows an average of ~2.8 dB better PSNR, but with a ~2.9 percent larger file size compared to the CPU-only implementation. These results demonstrate the performance–quality tradeoff and the tuning choices inherent in the two implementations.

Figure 8-7 shows the encoded video quality versus the encoding speed for this case study. It is evident that quality and speed scale among the different modes for both CPU-only and GPU-accelerated implementations, although the rate of scaling is different for the two implementations.

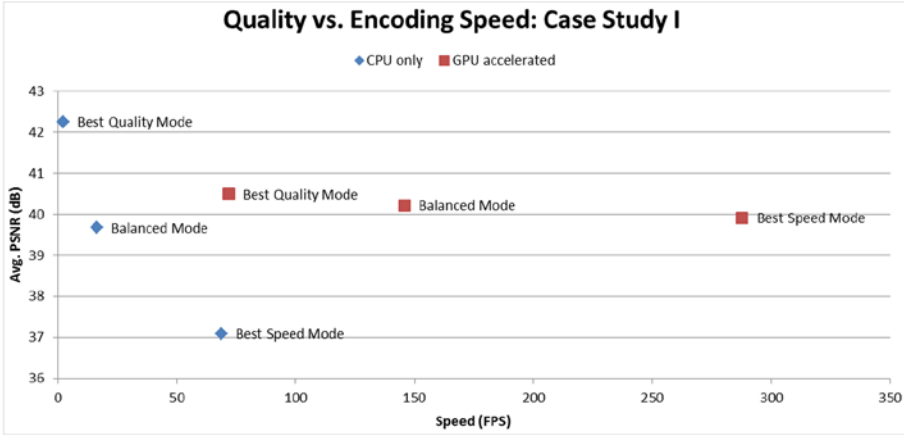

Figure 8-7. *Quality vs.encoding speed for case study I*

Case Study II

This second case shows another sample comparison of two encoding solutions in terms of performance and quality. A set of ten different video contents with varying complexities of motion and details are used. The video resolutions belong to the set {352×288, 720×480, 1280×720, 1920×1080}. Seven sets of video encoding parameters are used, providing a range between best quality and best speed. Encoding tests are carried out using two GPU-accelerated encoder implementations.

In this example, both encoding solutions operate on similar application program interfaces, such that parameter set 1 provides the best quality and parameter set 7 gives the best speed, although there are some differences between a parameter set for Encoder 1 compared to the same level of parameter set for Encoder 2. For example, parameter set 1 for Encoder 1 includes ⅛ pixel precision motion compensation and the use of trellis for mode decision, while Encoder 2 does not include these parameters in its parameter set 1. Some important parameters that are common to both two encoders are shown in Table 8-6.

Table 8-6. *Important Common Parameters between Encoder 1 and Encoder 2*

	PS1	PS2	PS3	PS4	PS5	PS6	PS7
8×8 transform	Yes	Yes	Yes	Yes	Yes	Yes	Yes
¼ pixel prediction	Yes	Yes	Yes	Yes	Yes	Yes	No
Adaptive search	Yes	Yes	Yes	Yes	Yes	No	No
Max references	10	8	6	5	4	2	2
Multiple prediction	Yes	Yes	Yes	Yes	P-picture only	No	No

Figure 8-8 shows the performance comparison between the two encoders in terms of FPS for each of the parameter sets. For both encoders, there are clear trends of improved performance with the progress of the parameter sets. However, the rates of improvement are different for the encoders. While Encoder 2 reaches the best performance of close to nine-fold faster than the real-time performance much more aggressively after parameter set 3, Encoder 1 displays a comparatively gradual rate of rise in performance as it steadily reaches about the same performance by tuning the parameters.

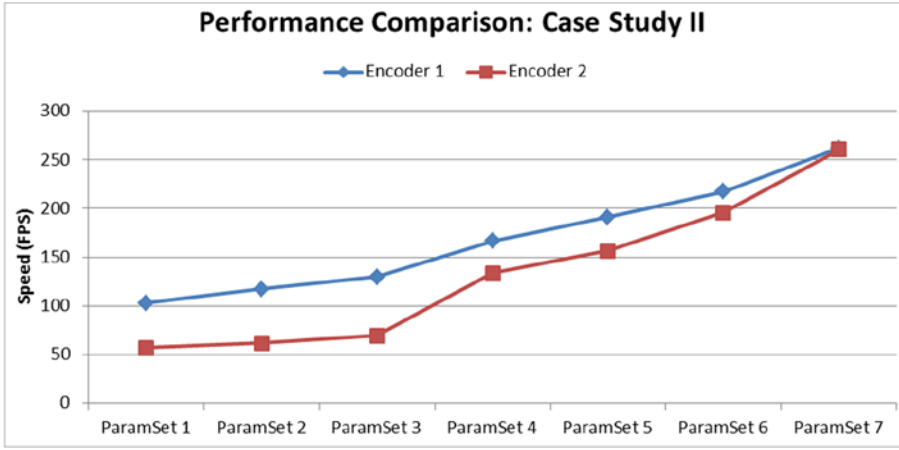

Figure 8-8. *Performance comparison of the two encoders*

Figure 8-9 shows a quality comparison between the two encoders in terms of BD-PSNR with respect to the parameter set 7 of Encoder 2. Again, clear trends of gradually lower quality are observed for both encoders. For Encoder 2, tuning the parameters can yield up to ~0.45 dB gain in BD-PSNR, while Encoder 1 reaches a level of ~0.47 dB quality gain. However, for Encoder 1, the mid-levels of parameter tuning do not show significant quality differences. Noticeable quality improvement for Encoder 1 happens between parameter sets 7 and 6, and between parameter sets 2 and 1.

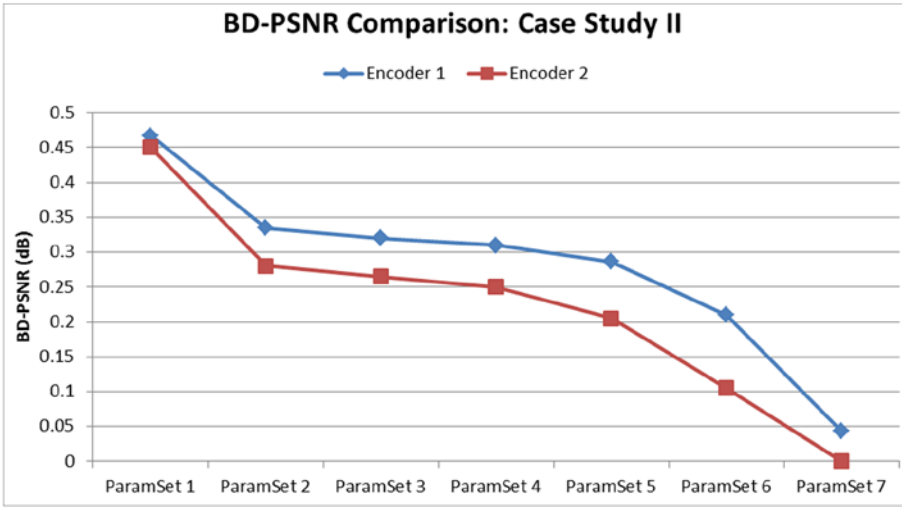

Figure 8-9. *Quality comparison of the two encoders*

Figure 8-10 shows quality versus encoding speed for the second case study. In general, for all sets of parameters, Encoder 1 provides better quality for a given encoding speed compared to Encoder 2. It is also clear that parameter set 1 truly represents the best quality mode, while the set 7 represents the best speed for both encoders. For Encoder 1, parameter set 6 appears to be the most effective, as it provides the largest quality difference (~0.1 dB BD-PSNR), but at the same time it also provides over 11 percent improvement in encoding speed compared to Encoder 2.

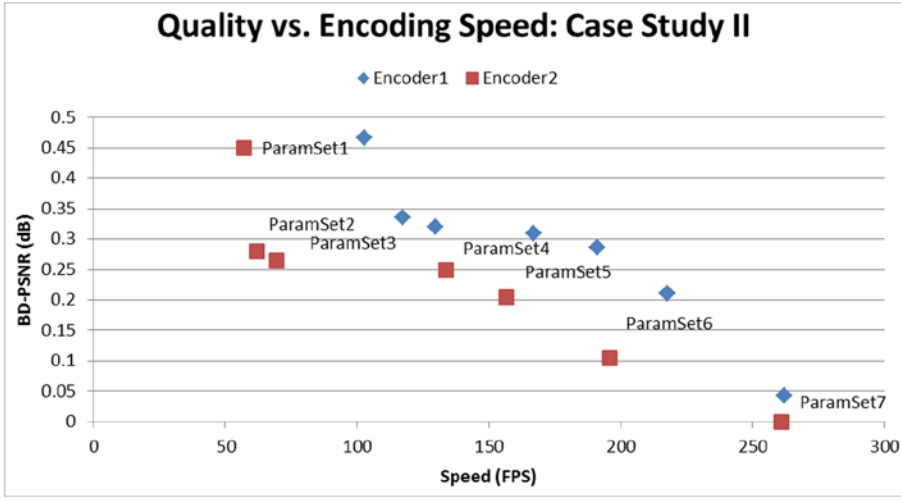

Figure 8-10. *Quality vs. encoding speed for case study II*

The Power–Quality Tradeoff

Noise is one of the most critical problems in digital images and videos, especially in low-light conditions. The relative amount of brightness and color noise varies depending on the exposure settings and on the camera model. In particular, low-light no-flash photo- and videography suffers from severe noise problems. The perceptual quality of video scenes with chroma noise can be improved by performing *chroma noise reduction*, alternatively known as *chroma denoise*. A complete elimination of brightness or luma noise can be unnatural and the full chroma noise removal can introduce false colors, so the denoising algorithms should carefully adapt the filtering strength, depending on the input local characteristics. The algorithm should represent a good tradeoff between reduction of noise and preservation of details. An example of a GPU-accelerated implementation of a chroma denoise filter, as a video processing capability, is typically available as an image-enhancement color processing (IECP) option offered by the Intel processor graphics.

To demonstrate the power–quality tradeoff, we present a case study of chroma denoise filtration. While playing back a video, the chroma denoise filter detects noise in the two chroma planes (U and V) separately and applies a temporal filter. Noise estimates are kept between frames and are blended together, usually at 8-bit precision. As the the GPU-accelerated chroma denoise typically provides sufficient performance for real-time processing, on modern processor platforms the performance is not normally a concern. However, although the visual quality is expected to improve, the additional operations required by the chroma noise reduction filter means that extra power is consumed. Therefore, this case study illustrates a tradeoff between power use and quality.

Case Study

This example uses a third-generation Intel Core i7 system with CPU frequency 2.7 GHz, turbo frequency up to 3.7 GHz, 45 W TDP, and graphics turbo frequency up to 1.25 GHz. The screen resolution is set to be 1920×1080, the same as the resolution of the video content. The balanced OS power policy is used, and the operating temperature is kept at 50°C, which is typical with CPU fans as the cooling system. Two workloads are employed, consisting of playback of an AVC encoded and a VC-1 encoded Blu-ray disc along with chroma denoise filter. Note that the VC-1 encoded content has much higher scene complexity compared to the AVC encoded content.

Figure 8-11 shows effect of the chroma denoise filter on the package power. An average of ~7 percent additional power, up to ~0.48 Watts, is consumed owing use of the chroma denoise filter. This is a significant penalty in terms of power consumption.

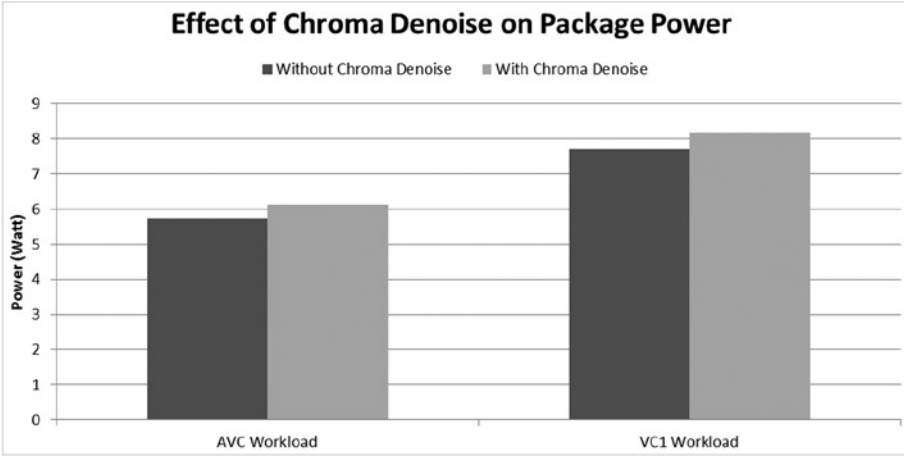

Figure 8-11. *Effect of chroma denoise filter on package power*

Figure 8-12 shows the effect of the chroma denoise filter on the combined CPU and GPU activity. From the activity point of view, there's an average of ~8.5 percent increase owing to the chroma denoise. This corresponds well with the increase in power consumption, and also represents a substantial increase in the time during which the processor is busy.

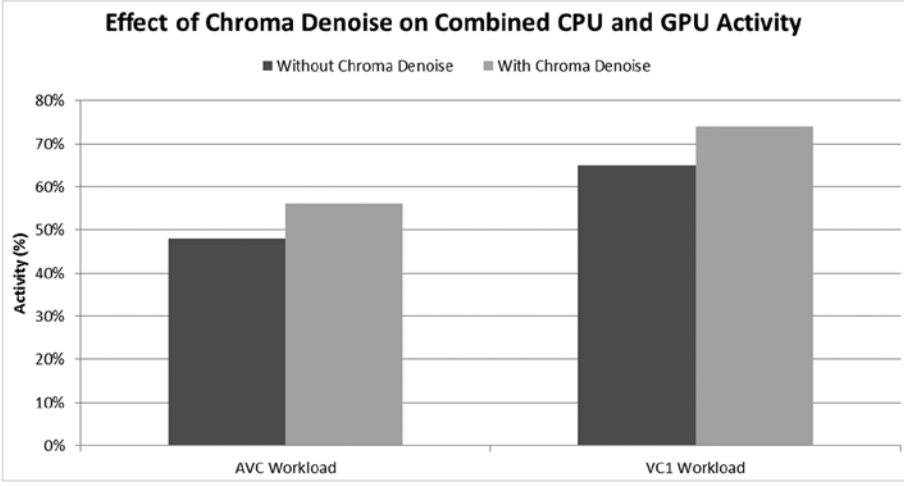

Figure 8-12. *Effect of chroma denoise filter on combined CPU and GPU activity*

Figure 8-13 shows the effects of the chroma denoise filter on visual quality, in terms of PSNR. Although an average of ~0.56 dB improvement is seen in PSNR, the perceived impacts on visual quality for these workloads are small. Note that the absolute PSNR value for the VC1 workload is about 4 dB lower than the AVC workload—this is due to the inherent higher complexity of the VC-1 encoded video content compared to the AVC encoded content.

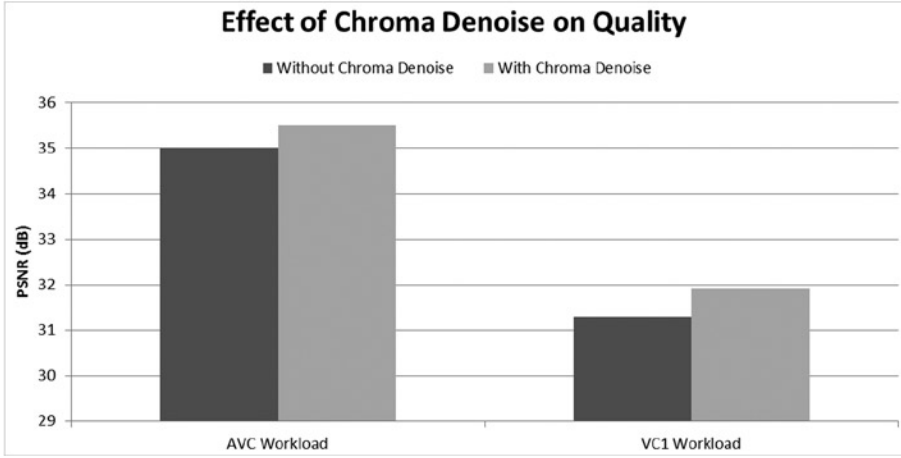

Figure 8-13. *Effect of chroma denoise filter on perceived visual quality*

Figure 8-14 shows the power–quality tradeoff for the chroma denoise filter. Although improved PSNR is observed, such improvement comes at the expense of substantially increased power consumption. Therefore, on power-constrained platforms, this option should be carefully considered. It is possible, for example, that upon detecting a lower level of battery availability, a power-aware playback application would automatically turn off the optional chroma denoise so as to save power.

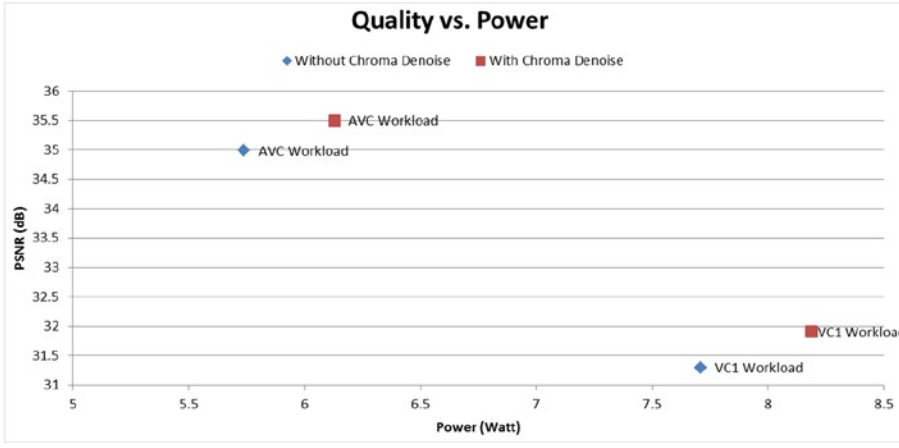

Figure 8-14. *Power–quality tradeoff for the chroma denoise filter*

Summary

In this chapter we discussed how tradeoff analysis works and we provided four practical examples. While a tradeoff analysis can involve consideration of many different dimensions, we focused on performance, power, and quality, which are the foremost criteria for success in today's low-power computing devices. While power consumption defines the achievable battery life, it joins highest encoding speed and best visual quality as the most desirable features of contemporary video coding solutions.

We discussed encoding parameter tuning and examined some optimization strategies. Further, we offered case studies that embodied performance–power, performance–quality, and power–quality tradeoffs. These case studies reflect several different points of view and help clarify the methodologies commonly used in such analyses.