Power Versus Quality Trade-offs for Adaptive Real-Time Applications

Andrew Nelson[1], Benny Akesson[2], Anca Molnos[1], Sjoerd te Pas[2], Kees Goossens[2]
[1]Delft University of Technology, Delft, The Netherlands
[2]Eindhoven University of Technology, Eindhoven, The Netherlands

***Abstract*—Electronic devices are expected to accommodate evermore complex functionality. Portable devices, such as mobile phones, have experienced a rapid increase in functionality, while at the same time being constrained by the amount of energy that may be stored in their batteries. Dynamic Voltage and Frequency Scaling (DVFS) is a common technique that is used to trade processor speed for a reduction in power consumption. Adaptive applications can reduce their output quality in exchange for a reduction in their execution time. This exchange has been shown to be useful for meeting temporal constraints, but its usefulness for reducing energy/power consumption has not been investigated.**

In this paper, we present a technique that uses existing DVFS methods to trade a quality decrease for lower power/energy consumption through an intermediary reduction in execution time. Our technique achieves this while meeting soft and/or hard time/energy/power constraints. We demonstrate the applicability of our technique on an adaptive H.263 decoder application, running on a predictable hardware platform that is prototyped on an FPGA. We further contribute an experimental evaluation of the H.263 decoder's scalable mechanisms, in their ability to trade quality for temporal/energy/power. From experimentation, we show that our quality trading technique is able to achieve up to a 45% increase in the number of frames decoded for the same amount of energy, in comparison to frequency scaling alone, but with a quality reduction of up to 22dB Peak Signal-to-Noise Ratio (PSNR).

***Index Terms*—Low-power design, Real-time systems, Embedded Systems**

I. Introduction

Energy and power constraints are an ever growing concern for designers and users of electronic devices [1]. Gadgets, such as mobile phones and tablet computers, that depend on battery power, are now ubiquitous in everyday life. Reducing energy and power consumption is desirable to increase the time between charges and to decrease the temperature of the device.

Dynamic Voltage and Frequency Scaling (DVFS) [2] is one approach that is commonly applied in order to reduce the power consumption of electronic systems. DVFS techniques lower the operating voltage and frequency, saving power at the expense of an extended execution time. For real-time applications, such as video and audio decoders, DVFS techniques have been demonstrated that lower the frequency of the application's execution while not violating the application's real-time requirements [2]–[4]. This is achieved by calculating what extended execution time, and hence frequency, is acceptable within the application's given temporal constraint. By decreasing the application's execution time through quality scaling, these existing DVFS mechanisms exchange the reduction in execution time for a reduction in power consumption.

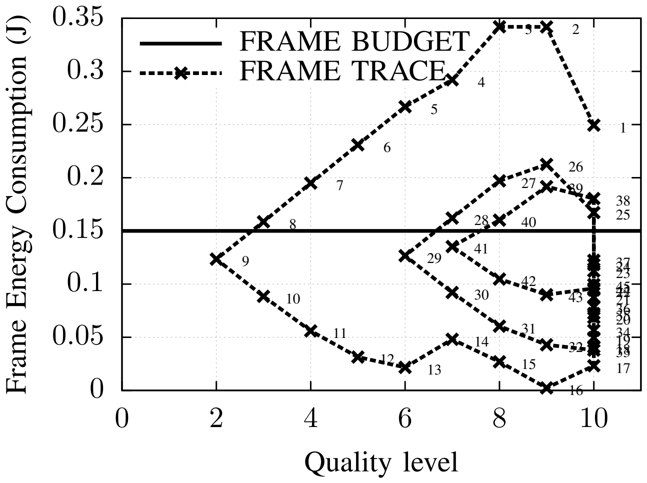


Figure 1. Adaptive H.263 decoder per frame trace of quality-level scaling to meet the indicated average energy target.

Adaptive applications [5] may change their execution profile at run-time, enabling, e.g. a slower exact algorithm to be substituted for a faster approximate algorithm at the cost of output quality. For example, in [6], [7], it was shown for an MPEG2 video decoder that output quality may be sacrificed to meet timing constraints by adapting the application.

Little work has been carried out to investigate using adaptive applications with quality-scalable algorithms to trade quality in order to meet temporal and energy requirements. [8] fulfils these specific criteria, by using a relatively complex run-time technique to maximise the quality level within given temporal and energy budgets. We propose a low-complexity technique, with a relatively small processing and energy overhead. Figure 1 shows how an adaptive H.263 decoder [9], [10], using our technique, can scale the quality of its algorithms in order to meet an average energy-per-frame budget. This is achieved by simply decreasing the quality level whenever the video is consuming more energy per frame than budgeted, and similarly increasing the level when it consumes less. Playing a video on a mobile phone is an example use case that can benefit from such a trade-off. If, for instance, a user wanted to display a video right until the end, but the phone's battery does not contain sufficient charge for this, and also retain enough energy to make a possible emergency phone call. Quality scaling for power reduction enables the user to have the option to watch the entire video at lesser quality, while not exceeding its allocated energy budget. This enables the user to retain some energy in the battery to make their potential emergency phone call.

In this paper, we contribute:

1) An experimental analysis of the adaptive mechanisms of an H.263 decoder [10] in regards to output quality and the amount of work required to produce a frame of decoded video.
2) A low-complexity run-time technique to trade a decrease in quality for a decrease in energy, via common DVFS techniques, in order to meet mixed criticality temporal/energy/power constraints.

By building on common DVFS techniques we ensure that our technique has wide applicability and generality. Our technique uses decoupled DVFS and quality scaling policies, aiding integration into platforms with existing DVFS policies, while still producing good combined results.

We demonstrate our technique for an adaptive H.263 decoder application mapped onto the existing CompSOC platform [11]. We analyse the effectiveness of our technique for the use case of scaling quality in order to achieve a particular number of decoded frames from an initial energy budget. We show that our technique works with both soft and firm real-time DVFS techniques, and that the DVFS technique does not significantly affect our technique's ability to trade a decrease in quality for a decrease in energy consumption. From experimentation on a single processing core, we show that our quality-scaling technique is able to extend the number of video frames decoded by up to 45% for the same energy budget, over using the DVFS technique on its own, in exchange for a quality reduction of up to 22dB of PSNR.

The rest of this paper is structured as follows. In the next section we give an overview of the current related work to the topic of this paper. We follow this in Section III with a description of pertinent background information before describing our technique in detail in Section IV. In Section V we provide an experimental analysis of our technique. We end this paper by drawing conclusions from our work in Section VI.

II. Related Work

In this section, we consider work that is related to our proposed technique. Our technique depends on adaptive applications that can exchange output quality for a reduction in their execution time. The term quality is application dependent, and even within an application domain may have different meanings, e.g. for video decoders quality could (non-exhaustively) refer to the number of deadline misses and/or the reproduction quality of the video frames using the PSNR metric. [5] evaluates the notion of adaptive applications in general. [12] looks at application adaptivity in terms of resource mapping rather than scalable algorithm complexity. [13] uses scalable algorithm complexity to adapt applications to improve hardware yield through process variation. While the applications described in these papers are adaptive, they do not have the correct sort of complexity scaling adaptivity for our technique.

[14], [15] presents dynamic QoS resource management where adaptive applications, with quality-scalable algorithms, are used in combination with resource budgeting in order to meet real-time requirements. [6], [7] demonstrate how an adaptive MPEG2 decoder may be used to trade quality in order to meet real-time requirements in consumer terminals. In [6], [7], quality decisions are made off-line and are fixed for the type of terminal. Similarly, [16], [17] show how quality scalable video algorithms may be applied for medical X-ray imaging on multi-application, multi-core architectures. These techniques demonstrate the applicability of adaptive application quality scaling to meeting temporal requirements, but do not use the scaling mechanisms to achieve an energy/power reduction.

In [18] adaptive MPEG2 DCT/IDCT algorithms are presented that can trade-off quality for an energy reduction. The work in [18] only considers the energy reduction due to the number and type of operations performed, and not the energy that can be subsequently be saved through DVFS while still meeting timing constraints. Similarly, [19] demonstrates a run-time technique to select appropriate encryption algorithms to meet real-time requirements, but also does not consider energy reductions through DVFS, in contrast to our technique.

DVFS is a commonly applied technique, that decreases the system's, operating voltage and frequency. [2]–[4] and many other works demonstrate how DVFS may be used with real-time applications, enabling a power reduction while meeting the application's temporal constraints. These techniques lower the operating frequency, stretching the application's execution time within the context of the temporal constraint. DVFS techniques can also be used to meet power and thermal constraints, as demonstrated in [20].

In [21], a run-time technique is proposed that trades application-level quality-of-service for power reduction, on multi-core multimedia platforms. They present their technique for an MPEG2 decoder, using deadline miss rates as their quality metric. A run-time linear programming technique is used to find optimal task mappings on an MPSoC, considering core performance variations due to production process variability. As opposed to this paper, the technique presented in [21] does not consider quality scaling of adaptive applications, in order to achieve an energy/power reduction.

In [8], task scheduling and DVFS levels are selected in order to maximise quality within a given temporal and energy budget. The problem is formulated as an integer linear program that is solved at run-time after every task execution. From their own evaluation of their technique, solving the linear program costs thousands of cycles each time, adding both a processing and energy overhead. An evaluation of the energy/power performance of the technique is not provided in [8].

In this paper, we describe a low-complexity (hence low overhead) technique to trade a quality reduction for a re-

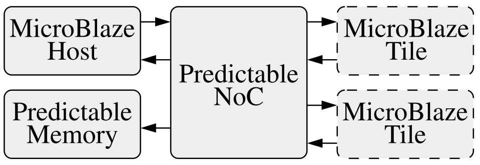


Figure 2. CompSOC predictable hardware platform.

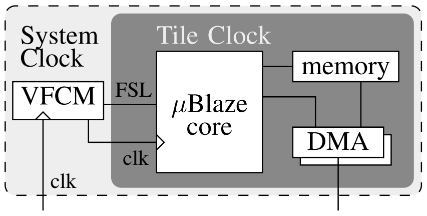


Figure 3. MicroBlaze (μBlaze) based processing tile.

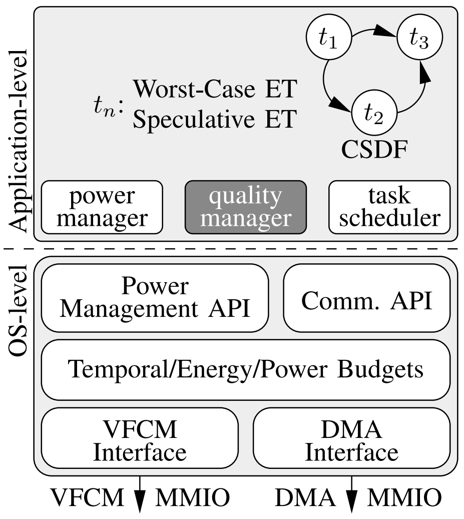


Figure 4. CompOSe and application structure.

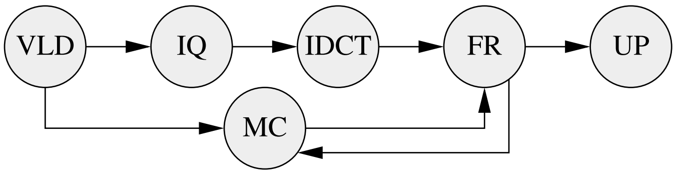


Figure 5. H.263 decoder application task graph.

duction in energy/power consumption, while meeting real-time throughput requirements. We demonstrate this for an adaptive H.263 decoder application [10], which has scalable complexity algorithms. We use run-time complexity scaling to satisfy mixed criticality temporal/energy/power constraints, at the expense of quality. Energy and power reductions are made by trading temporal savings from quality scaling for voltage and frequency reductions using common run-time DVFS techniques.

III. Background

We proceed now to present the relevant background information, required by our quality scaling technique. In order to be able to guarantee that we meet real-time constraints, we use the CompSOC predictable platform, which we describe in Section III-A. We use an adaptive H.263 decoder as the running example application throughout this paper, which we describe in Section III-B.

A. Predictable Platform

In order to meet temporal constraints, our technique requires a DVFS capable real-time hardware platform. For this purpose we use the CompSOC predictable MPSoC, as illustrated in Figure 2. The CompSOC platform is a NoC-centric tile-based platform. Processing tiles consist of a MicroBlaze processing core, with local instruction and data memories, as illustrated in Figure 3. DVFS is enabled using the hardware Voltage and Frequency Control Module (VFCM). The VFCM module models frequency scaling through clock sub-sampling. Voltage scaling is taken into account using a frequency to power model, assumes a minimum voltage for a given frequency, as per [3]. Tiles are also equipped with DMAs for inter-tile communication, allowing the parallelisation of computation and communication. Inter-tile communication is carried out using the predictable Æthereal NoC. The CompSOC platform enables the calculation of worst case computation and communication timings, making the platform predictable.

On top of the CompSOC hardware platform we run the CompOSe OS, as illustrated in Figure 4. This OS executes applications that are structured as task graphs, such as Cyclo Static DataFlow (CSDF). The task graphs are annotated with worst case timings, that when coupled with the predictable nature of the hardware, enables the implementation of DVFS policies that can lower the voltage and frequency, while maintaining an applications real-time requirements.

The CompOSe OS provides a power-management API interface that provides run-time budgeting data and functions, in order to implement DVFS policies as power-management functions. A DVFS policy that is implemented in multiple DVFS capable real-time systems [2], [3], involves observing slack in the applications schedule, due to static or dynamic timing variations, and then using this observed slack to perform DVFS. For firm real-time applications, by knowing the application's worst-case timings and only using slack that has been observed the frequency can be scaled conservatively. Using the CompOSe OS we are able to implement this and other DVFS policies.

B. Adaptive Application

In order to demonstrate our technique we apply it to an adaptive baseline H.263 video decoder application [9]. The H.263 application decodes the input video stream as illustrated in Figure 5. The compressed stream first undergoes Variable Length Decoding (VLD). The resultant stream consists of macroblocks, with each macroblock containing frequency encoded YUV information for an 8×8 block of pixels.

Macroblocks belong to either an I-frame or a P-frame. I-Frames contain the information to reconstruct all the pixels

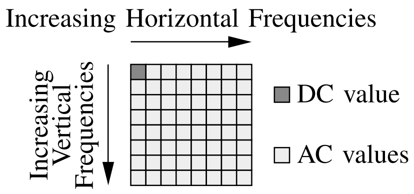

Increasing Horizontal Frequencies
Increasing Vertical Frequencies
DC value
AC values

Figure 6. 8 × 8 macroblock.

of the encoded frame. These frames are reconstituted through Inverse Quantisation (IQ), Inverse Discrete Cosine Transform (IDCT), and finally Frame Reconstruction (FR). P-frames do not contain the encoded version of the entire frame. Instead these frames contain Motion Compensation (MC) information, that groups pixels with vector translations, allowing the frame to be reconstructed from the previously reconstructed frame. The reconstructed I/P-frames are Up-scaled (UP) to fit the allocated display area.

An adaptive H.263 decoder [10] contains parametrised adaptive functions in the application that allow us to decrease the decoder's execution time in exchange for a reduction in the decoder's output quality. The adaptive H.263 decoder used here contains the following two adaptabilities:

1) Parametrised quantity of decoded AC values in a macroblock.
2) Parametrised up-scaling complexity.

A frequency domain macroblock is encoded using a Discrete Cosine Transform (DCT) [9]. An 8×8 value macroblock of this type consists of a single DC value, that represents the average value for the macroblock, and 63 AC values, as illustrated in Figure 6. As shown in this 2 dimensional representation of the macroblock, the further in each dimension an AC value is from the DC value, the higher the frequency is that it represents in that dimension. By selectively ignoring AC values, the time taken for the IDCT task of the decoder may be decreased, in exchange for a reduction in reproduction quality of the spatially encoded macroblock. The encoding process places less value on higher frequency information in the macroblock, due to human perception. Similarly the adaptable function in the decoder allows scaling of the amount of processed AC values, ignoring AC values at the higher end of the frequency spectrum first.

Parametrised up-scaling complexity is achieved by a similarity-threshold parameter passed to a bi-linear interpolation algorithm. If the two pixels under comparison are similar to within the threshold value, then no interpolation takes place. In this eventuality one of the compared values is simply reproduced. If the two pixels are suitably dissimilar, bi-linear interpolation is performed. By relaxing the similarity threshold, a reduction in execution time is achieved at the expense of the reproduction quality of the final image.

IV. Quality for Power Reduction

We continue by describing how our temporal and power constrained quality scaling technique works in general. We further show how it may be applied in practice using the CompSOC platform and the adaptive H.263 decoder described in Section III.

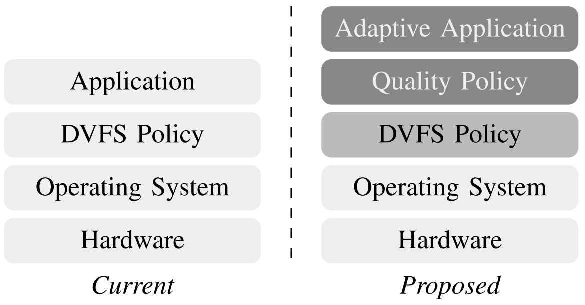


Figure 7. System hierarchy.

		Time	
		surplus	*deficit*
Energy	*surplus*	decrease frequency	increase frequency
	deficit	decrease frequency	increase frequency

(a) Power-management DVFS policy. (Regardless of energy/power budget constraint)

		Time	
		surplus	*deficit*
Energy	*surplus*	increase quality	decrease quality
	deficit	decrease quality	decrease quality

(b) Quality-management policy

Table I
Quality- and power-management policies, based on run-time temporal/energy/power budget information.

A. Applied in General

In order to achieve this we introduce a quality management layer between the application and existing DVFS capable system, as illustrated in Figure 7.

The quality-management layer enacts the application's quality policy. Example quality- and power-management policies are shown in Table I. The policies indicate the course of action that is taken by the quality- and power-management, in the context of run-time temporal and energy budget information. For simplicity, Table I describes the policies for coarse-grained budget information, i.e. whether the budgets are being under- or overused. At run-time, temporal and energy budgets may be underused, creating a budget surplus (i.e. slack), and they may also be overused, creating a budget deficit (i.e. running low in terms of energy, or late in terms of time).

The quality-management policy in Table Ib and the power-management policy in Table Ia are two independent polices. This allows the technique to be applied more easily to platforms that already have DVFS power-management policies. The quality- and power-management may also operate at different granularities, e.g. for the H.263 decoder, the quality may change every frame while the frequency changes every macroblock. This allows our technique to be more general and therefore more widely applicable, than by having a single policy.

Four scenarios are presented in Table I, covering the possible combined scenarios of temporal and energy budget under- and overuse. If the temporal budget has slack then the power-manager can lower the frequency and voltage conserving power, while still meeting the temporal requirement. If the temporal budget is showing that the application is running late, then the power-manager must increase the frequency and voltage to meet the temporal requirement. The increase in the frequency also means and increase in power consumption, and is therefore dependent on the slack in the energy/power budget. If the temporal and energy/power budgets are concurrently running deficits, then a conflicting situation arises whereby the system needs to increase the frequency in order to meet the temporal budgets constraint, while at the same time needing to lower energy/power consumption in order to meet the energy/power budgets constraint. In the policy in Table I, a decision is made so that if an increase in frequency is required to satisfy the temporal constraint, and the energy/power budget does not have slack, then the frequency is increased regardless of the energy/power budget constraint.

Scaling the quality of an adaptive application, such as the H.263 decoder described in Section III, enables a trade-off in quality for a reduction in execution time. This reduction in execution time can be used to meet temporal constraints, or traded through DVFS for a reduction in power consumption, in order to meet energy/power constraints. As illustrated in Figure 7, our technique provides a quality-management layer between the application and the existing DVFS policy. The quality-manager has access to the application's budgeting information, and may also use information regarding power-management policy. Table I shows an example quality policy for use with the quality-manager. In this policy the quality is decreased when the temporal or energy/power budgets are running a deficit while trying to meet their constraints. Quality may be increased again whenever all the budgets have slack.

Our technique permits mixed criticality temporal/energy/power constraints within an application. This means that applications using our technique may have concurrent temporal/energy/power constraints with mixed degrees of severity, e.g. for the use case of playing a video on a mobile phone, the video play back is temporally soft real-time, while it's energy budget is a firm constraint, which subsequently results in a soft criticality power constraint per frame. A suitable policy needs to be selected depending on the criticality of the constraint. Conservative policies ensure that the temporal/energy/power constraint is always met, and are therefore used for firm real-time applications. This is achieved by using worst-case task timing/energy information when making quality-/power-management decisions. Speculative policies in our system use run-time observed average task timing/energy information when making quality-/power-management decisions, and are therefore used for soft real-time applications. Both guaranteed and speculative policies only use run-time observed budget slack, making the quality- and power-management a closed-loop control system.

B. Applied to the CompSOC Platform

We now proceed to explain how our technique may be used with a DVFS capable predictable platform, for an H.263 decoder adaptive application. For this we apply our technique to the CompSOC platform, running the CompOSe OS, as described in Section III. The CompOSe OS facilitates DVFS through the use of a user specified power-management function at the application-level, as illustrated in Figure 4. To fit our aim of using quality management with existing DVFS platforms, as is illustrated in Figure 7, we introduce a user specified quality-management function at the application-level of the CompSOC platform.

The quality-management function enacts the quality policy, such as the policy that is described in Table I. It achieves this by returning an enumerated quality-level to the application, that the scalable mechanisms use to scale the quality. In our implementation the enumerated quality-level is a Natural number from 1 to 10, with 1 being the lowest quality and 10 being the highest. The function is called from within the adaptive application. The frequency of calling the quality-manager function is a design decision. For the H.263 decoder, the quality-manager is called on the granularity of video frames.

As with the power-manager, the quality manager accesses run-time budget information via the power-management API. As such, the quality-manager has access to the same budgeting information as the power-management function. The H.263 decoder is a real-time application, and therefore a temporal budget for the application is maintained. An energy constraint is also placed on the H.263 decoder, specifying how much energy the application can maximally use to complete the video. This firm constraint for the entire video, translates into a soft criticality power constraint that is represented as an energy constraint per-frame. The relevant budgeting information, provided by the CompOSe power-management API, consists of the following:

- `time_budget` Amount of time in system time budgeted for n application graph iterations.
- `used_time_budget` Current time in system time used from the temporal budget.
- `energy_budget` Amount of energy budgeted for the entire application execution (i.e. for decoding the entire video).
- `power_budget` Amount of energy budgeted for the current time interval (i.e. video frame duration).
- `used_power_budget` Current energy used from the power budget, for the current time interval (i.e. video frame duration).

System time is the time measured in cycles, by counting the clock pulses of the unscaled system clock, that runs at the system's maximum frequency. There is a linear relationship between the progression of system time and wall time. The amount of temporal slack that is available in the budget is calculated by subtracting the `used_time_budget` from the `time_budget_constraint`.

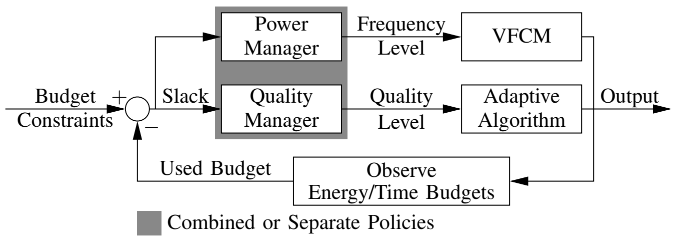


Figure 8. Quality-management control loop.

The power_budget_constraint is calculated by dividing the energy_budget_constraint by the number of frames in the video to obtain the power budget constraint, in terms of energy per frame. Similarly, the amount of energy slack that is available in the power budget is calculated by subtracting the used_power_budget from the power_budget_constraint. Using the slack values for the temporal and power budgets, the quality-manager is able to enact quality-management policy from Table I. For the adaptive H.263 decoder, the power-management function looks as follows:

```
quality_level h263_quality_manager(){
 time_slack   = getTimeBudgetConstraint();
 time_slack  -= getUsedTimeBudget();
 power_slack  = getPowerBudgetConstraint();
 power_slack -= getUsedPowerBudget();

 if(time_slack < 0 || power_slack < 0){
  quality_level = getCurrentQualityLevel() - 1;
 }else if(time_slack > 0 && power_slack > 0){
  quality_level = getCurrentQualityLevel() + 1;
 }

 return quality_level;
}
```

where get functions are used from the CompOSe power-management API to access the budget information. For this policy, if both temporal and power slack are positive, then the quality-level is raised. If either temporal or power slack is negative, then the quality level is lowered. In all other combinations, the quality-level is maintained at its current level.

For the adaptive H.263 decoder the quality- and power-management control loop, illustrated in Figure 8, is executed on a per frame granularity. The quality-manager is called at the start of every frame. The obtained quality-level is then passed to the application's adaptive functions, as per the control flow illustrated in Figure 8. Any observed reduction in execution time will be noted in the temporal budget. The power-management observes this reduction when it calculates the temporal slack. The power-manager changes the frequency according to the policy in Table I. If the H.263 decoder is running late with its temporal requirement, the frequency is set at its highest level. If the H.263 decoder is ahead of its temporal requirement, the power-manager lowers the frequency as much as possible within the constraint of the observed slack. Changing the frequency, modifies the observed temporal/energy/power consumption in the budgets, which in turn affects the next chosen quality-level by the quality-manager.

V. Experimentation

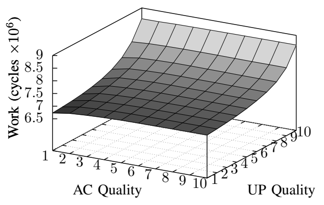
We continue by experimentally analysing the pertinent properties of our technique. We apply our technique to the adaptive H.263 decoder from Section III-B, running on the CompSOC platform from Section III-A. Our experimentation is carried out on an FPGA prototype of the CompSOC platform, using the Xilinx ml605 prototyping board. Using this platform, we contribute an experimental analysis of the adaptive H.263 decoder's scalable mechanisms, that are described in Section III-B. We investigate the relationship between the requested quality level and the output quality of the decoded video frame, measured as a Peak Signal to Noise Ratio (PSNR) in comparison to the reference frame, decoded at the highest quality settings. We also investigate the relationship between the requested quality level and the amount of work required to decode a single frame of video.

Following this, we evaluate our quality-scaling technique by providing an in depth analysis of its application to the use case of decoding a particular number of video frames for a given energy budget. Due to the decoupled nature of the power management (DVFS) and quality management policies described in Table I, we evaluate the pairing of the quality management with both conservative and speculative real-time power management policies.

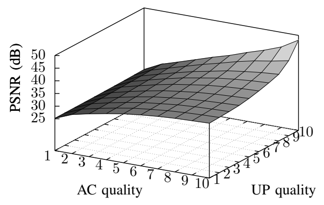
While our experimental results show absolute power and energy estimates, *we do not make any claims about the accuracy of our used power model*. The power model that we use is *for comparative purposes only*, enabling us to evaluate whether our technique provides an improvement in comparison to the same situation without our technique. Our processor power model is based on the power consumption estimate of the MicroBlaze processor, at 120MHz, for the ml605 board's virtex 6 FPGA. We obtained an estimate of 348mW using the Xilinx "Xpower Analyzer" tool, for a mapped and routed instance of a MicroBlaze processor on the FPGA. What is important for demonstrating the validity of our technique, is that by lowering the processor's operating frequency (and voltage to match) the processor's power consumption decreases. As such, we use a simple linear relationship between frequency and power, but emphasise that our technique also works for other monotonic frequency/power models, such as those with a quadratic or cubic relationship, as may be obtained from the parametrised model described in [3].

A. H.263 Scalable Quality Mechanisms

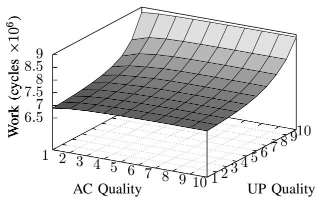
We start out experimentation by evaluating the H.263 decoder's scalable quality mechanisms. For our first experiment we evaluate the amount of work that has to be performed, in processor cycles, in order to produce a frame of decoded video at a particular requested quality-level. The H.263 decoder has two scalable quality mechanisms, as described in



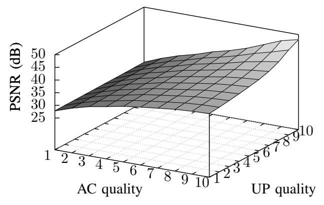
(a) Akiyo average work.



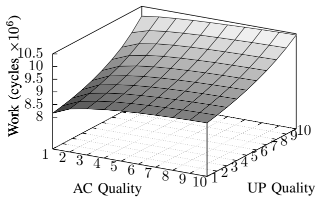
(b) Akiyo average PSNR.



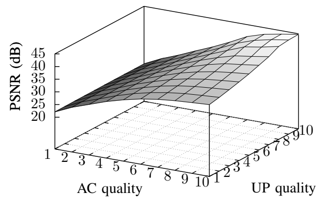
(c) Tree average work.



(d) Tree average PSNR.



(e) Bus average work.



(f) Bus average PSNR.

Figure 9. Quality mechanism performance, for the average of 30 decoded frames.

Section III-B, each having 10 discrete quality levels, giving 100 possible quality combinations that may be requested. In order to obtain a work measurement for each of these quality combinations, a video is decoded with the quality-level fixed at that combination for the duration of the decoding. In order to provide a single value per quality-level, we take the average work required to decode a video frame from the first 30 decoded frames.

Figures 9a, 9c and 9e show the work to produce a decoded frame against quality-level results for the decoded akiyo, tree and bus reference videos respectively. The resultant surfaces are not exactly the same due to the data dependent nature of the quality scaling mechanism's execution. From the graphs it can be seen that by decreasing either of the two quality scaling mechanisms, that there is a monotonic reduction in work that must be performed to decode a frame. This is a useful property of the quality-scaling mechanisms as it ensures that requesting a lower quality-level will not lead to extra work having to be performed.

From Figures 9a, 9c and 9e, it is apparent that the up-scaler quality mechanism produces a larger reduction in work needed to decode a frame, across its range of quality-levels, than the AC-values quality mechanism. The amount of work

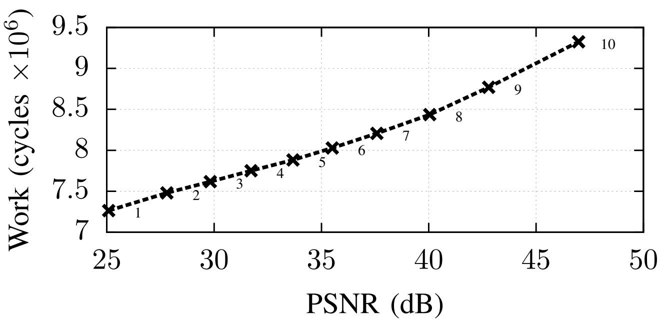


Figure 10. Work against PSNR for the 10 quality levels.

that can be saved by scalable algorithms is algorithm specific. It depends on the physical nature of the algorithm, or the minimum acceptable quality of output that is still useful. For each scalable algorithm, the degree of received quality across their scalable ranges will also vary. Since we are trading output quality for a reduction in work, and hence a reduction in energy consumption, it is important to evaluate the received quality-level (PSNR) from each mechanism in comparison to the requested quality-level, which leads us to our next experiment.

For our second experiment we evaluate the received quality, measured as Peak Signal to Noise Ratio (PSNR), in comparison to the requested quality. In order to obtain a PSNR measurement for each of these quality combinations, a video is decoded with the quality-level fixed at that combination for the duration of the decoding. In order to provide a single value per quality-level, we take the average PSNR of the first 30 decoded frames.

Figures 9b, 9d and 9f show the PSNR against quality-level results for the decoded akiyo, tree and bus reference videos respectively. As with the surfaces produced by exchanging a reduction in quality for a reduction in work, the resultant surfaces here are also not exactly the same due to the data dependent nature of the quality scaling mechanism's execution. Nevertheless, the results from the three input videos show a monotonic decrease in received quality-level as the requested quality-levels are decreased. Reducing the quality-level for the AC-value mechanism shows the largest quality reduction at its lowest quality value, with an $\approx$10 dB difference in comparison to the up-scaler mechanism at its lowest quality value.

Due to the data dependent nature of the required work and eventual quality (PSNR) of both adaptive mechanisms, it is not possible to derive a generic linear path through all 100 possible quality levels that provides a monotonic trade-off between quality and work. As such, for the rest of our experimentation we scale both mechanisms equally, producing 10 possible quality levels that provide the monotonic trade-off we require, as shown in Figure 10. From this graph we can see that on average a work reduction of 2×10^6 cycles is achievable for a quality reduction of 22dB PSNR.

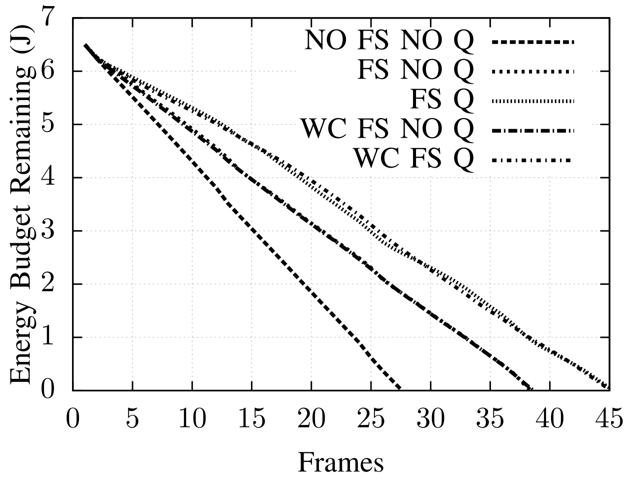


Figure 11. Remaining energy budget.

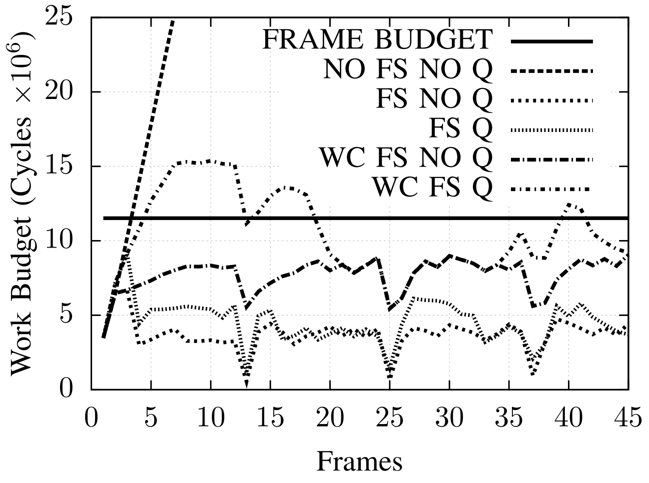


Figure 12. Remaining work budget after each frame.

B. Practical Application

Having evaluated the H.263 decoder's quality-scalable mechanisms, we proceed to evaluate the practical application of these mechanisms. It is described in Section IV-B how an adaptive application may be executed on the CompSOC platform, using the platform's budgeting mechanisms in order to make quality management decisions. We evaluate the practical usage of the quality scaling mechanisms for the use-case that is described in Section I. Given a particular energy budget, such as the remainder of the energy in a devices battery, we evaluate the applicability of quality-scaling in combination with frequency-scaling in order to meet a target number of decoded video frames.

Our experimental evaluation consists of decoding a video five times:

NO FS NO Q no frequency scaling or quality scaling.

FS NO Q speculative frequency scaling no quality scaling.

FS Q speculative frequency scaling, and quality scaling.

WC FS NO Q conservative frequency scaling no quality scaling.

WC FS Q conservative frequency scaling, and quality scaling.

The decoder is given an energy budget of 15J to decode 45 video frames, while maintaining a real-time throughput requirement of 10 frames per second. The depletion of the energy budget during the three different runs can be seen in Figure 11. Without frequency scaling and quality scaling (NO FS NO Q) the video is decoded at the processor's maximum frequency. As can be seen in Figure 11 running at the processor's maximum frequency causes the energy budget to deplete at a relatively fast rate, causing the energy budget to be depleted by the 28th frame.

Enabling frequency-scaling (FS NO Q) allows the processor to run at lower frequencies, thereby consuming energy at lower rates. The power-management policies described in Section IV-B scales the processor frequency while meeting the application's real-time requirements. Figure 12 shows that given a per-frame work budget suitable to achieve 10fps on a processor with a maximum frequency of 120MHz, that the power-management is able to scale the frequency, as shown in Figure 12, while meeting the real-time requirement.

In comparison to the situation with no frequency-scaling (NO FS NO Q) in Figure 12, by running at maximum frequency all the time the H.263 decoder continuously under uses its work budget, causing it to continuously accumulate. In Figure 11 it can be seen that by enabling frequency-scaling (FS NO Q) that the energy budget now stretches to the 37th frame, which is a 32% improvement but is still short of the 45 frame target.

The frequency scaling is done in accordance with either a speculative or conservative power-management policy. The conservative policy (WC FS NO Q) is guaranteed to meet the throughput requirement by scaling the frequency assuming the worst-case task execution times, whereas the speculative policy (FS NO Q) assumes the average task execution time from the previous frame. The work budget is accumulative, meaning that it retains slack between frames. If the conservative policy runs faster than the speculative policy in a particular frame then the conservative policy has more slack to use for frequency scaling in the next frame. Figure 13 shows that on average the conservative and speculative policies use similar frequencies, which in turn translates into similar energy consumption, as is apparent from Figure 11 where the FS NO Q and WC FS NO Q lines almost completely overlap.

Frequency-scaling alone is not able to reduce the energy consumption rate further without affecting the video's decoded frame rate. By enabling quality-scaling in conjunction with frequency-scaling (FS Q), the 45 frame target is met within the given energy budget, as shown in Figure 11. With the use of quality-scaling and frequency-scaling (FS Q), the same initial energy budget lasted for 22% more frames than frequency-scaling alone (FS NO Q), and for 60% more frames than without frequency- and quality-scaling (NO FS NO Q).

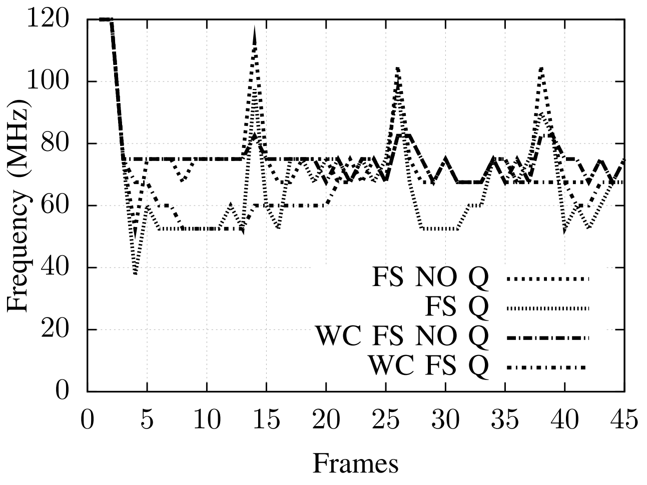


Figure 13. Processor frequency-level per frame.

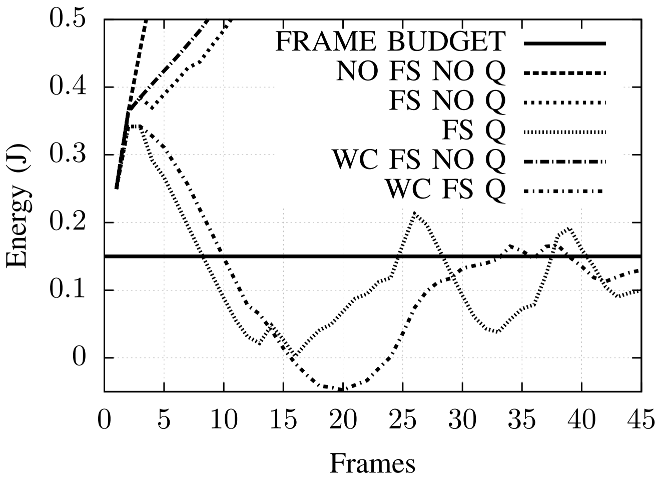


Figure 15. Energy budget consumption per frame.

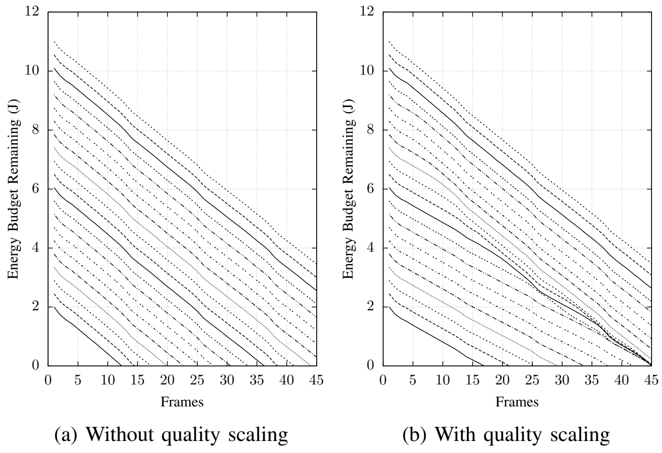


Figure 14. Energy budget depletion from various starting points, while decoding H.263 video with a soft 45 frame minimum requirement.

We run the experiment again with varying starting energy levels, with and without quality scaling, but still with a minimum frame requirement of 45 frames, producing the figures shown in Figure 14. Figure 14a demonstrates the battery depletion for frequency scaling without quality scaling. The power-management policy described in Table Ia scales the frequency to meet temporal requirements but does not take energy requirements into account. Figure 14b demonstrates the battery depletion with both frequency scaling and quality scaling enabled. In contrast to the power-management policy, the quality-management policy described in Table Ib tries to meet energy requirements. This can be seen in Figure 14b as the quality-manager tries to make the energy budget last for 45 decode frames. Some starting energy budgets are too low and cannot stretch to 45 frames, even at the lowest quality setting, and other starting energy budgets are so large that the 45 frame target is met without any quality scaling. A funnel shaped region exists in Figure 14b where run-time adjustments made by the quality manager to the quality level are effective to meet the 45 frame requirement.

As described in Section IV-B, the total energy budget is divided among the number of frames to create a per frame energy budget. Figure 1 shows a per frame trace of how the quality manager adjusts the quality level in respect to this budget, for the policy described in Table Ib. Whenever there is an energy budget surplus, represented by the trace being below the budget line, the quality is increased one quality-level per frame. Whenever there is a deficit, represented by the trace being above the budget line, the quality-level is decreased by one level per frame. With quality scaling enabled the policy keeps the energy per frame close to the budget. This is sufficient to meet the energy budget's soft real-time requirement.

The outcome of the five different experimental runs are shown in relation to the per frame energy budget in Figure 15. Both runs with quality scaling enabled, FS Q and WC FS Q, are shown to keep the energy consumption close to the budgeted amount. The quality levels that they use to achieve this are shown in Figure 16. From this graph it can be seen that the quality management with the conservative frequency scaling policy produces the highest quality for more frames, but also reaches the lowest quality level of the two runs. The quality scaling policy does not take into account the power-management's speculative or conservative real-time policy, only if the work budget has a surplus or a deficit. Even though the work budget continuously had a surplus for both runs with quality scaling enabled, as can be seen in Figure 12, the difference between the speculative and conservative polices affected the frequency levels and hence the rate of energy consumption. As per Figure 8, this in turn affects the chosen quality level by the quality management, which affects the work required to decode a frame, leaving more/less slack and energy for the next quality- and power-management decisions.

As is explained in Section IV-A, a reduction in quality-level translates into a reduction in work that needs to be performed in order to decode the frame. This in turn translates into

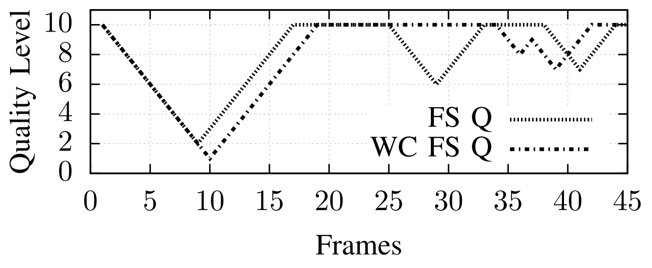


Figure 16. Requested quality-level.

an accumulation of extra slack in the work budget, thereby enabling lower frequencies to be used to decode the frame while still meeting the application's throughput requirement. This effect can be seen in Figure 13 where the frequency-scaling and quality-scaling combined (FS Q) is able to run more of the time at lower frequencies than without quality-scaling (FS NO Q). This is achievable while meeting the video decoder's throughput requirement, as can be seen in Figure 12. While the underlying conservative or speculative real-time DVFS policy has an affect on quality management decisions the overall effect on energy consumption, as seen in Figure 11, is not appear very significant from our experimentation. As such, we have shown that our quality management technique is suitable for use with conservative and speculative real-time DVFS policies, but that there is no significant advantage of using one over the other in relation to energy/quality trade-offs.

VI. Conclusion

The energy and power savings that can be made using quality scaling, with adaptive applications, are application and platform dependent. We show how these scaling mechanisms may be applied in a practical context for an H.263 adaptive real-time application executing on an existing MPSoC platform. By using independent power- and quality-managers our technique is able to be integrated more easily onto platforms with existing real-time DVFS power-management techniques, while still achieving the quality/energy trade-off we require.

We show that our technique can be used with soft and firm real-time DVFS mechanisms. From experimentation there does not appear to be a significant difference in their ability to work with our quality scaling technique, with both techniques performing very similarly. Our quality scaling technique is shown to work for mixed criticality temporal/energy/power constraints. We demonstrate this for the practical application of scaling the output quality of the H.263 video decoder in order to meet a given energy budget for a particular number of decoded video frames. Through experimentation, using an actual platform instance executing on an FPGA prototyping board, we show that quality-scaling enables the same level of energy budget to be used to decode more frames than with frequency-scaling alone. From our experimentation we show that the same level of energy budget can be used to decode up to 45% more frames when using quality-scaling, than frequency scaling alone, but at a cost of up to 22dB PSNR.

Acknowledgements This work was partially funded by projects EU FP7 288008 T-CREST and 288248 Flextiles, Catrene CA501 COMCAS and CA104 Cobra, and NL STW 10346 NEST.

References

[1] ITRS, "System drivers," 2011, http://www.itrs.net.

[2] P. Pillai *et al.*, "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems," ser. SOSP, 2001.

[3] R. Jejurikar, "Leakage Aware Dynamic Voltage Scaling for Real-time Embedded Systems," in *DAC*, 2004.

[4] A. Nelson *et al.*, "Composable Power Management with Energy and Power Budgets per Application," in *SAMOS*, 2011.

[5] P. McKinley *et al.*, "Composing Adaptive Software," *Computer*, 2004.

[6] C. Wüst *et al.*, "QoS Control Strategies for High-Quality Video Processing," *Real-Time Systems*, 2005.

[7] R. Bril *et al.*, "Multimedia QoS in Consumer Terminals," ser. SPS, 2001.

[8] H. Yu *et al.*, "Quality-Driven Dynamic Scheduling for Real-Time Adaptive Applications on Multiprocessor Systems," *IEEE Transactions on Computers*, no. PrePrints, 2012.

[9] *Recommendation H.263*, ITU-T, 2005, http://www.itu.int/rec/T-REC-H.263-200501-I.

[10] S. te Pas, "Quality versus energy trade-off for real-time applications on a composable MPSoC," Master's thesis, TU Eindhoven, 2012.

[11] B. Akesson *et al.*, "Composability and Predictability for Independent Application Development, Verification, and Execution," in *MPSoC: Hardware Design and Tool Integration*, 2010.

[12] R. Vanegas *et al.*, "QuO's Runtime Support for Quality of Service in Distributed Objects," ser. Middleware, 2009.

[13] A. Pant *et al.*, "Software Adaptation in Quality Sensitive Applications to Deal with Hardware Variability," in *GLSVLS*, 2010.

[14] G. Valls *et al.*, "Real-time reconfiguration in multimedia embedded systems," *IEEE Transactions on Consumer Electronics*, 2011.

[15] ——, "A dual-band priority assignment algorithm for dynamic QoS resource management," *Future Generation Computer Systems*, 2012.

[16] R. Albers *et al.*, "QoS management of dynamic video tasks by task splitting and skipping," in *ESTIMedia*, 2009.

[17] R. Albers, "Modeling and control of image processing for interventional X-ray," Ph.D. dissertation, Eindhoven University of Technology, 2010.

[18] R. Henning *et al.*, "An Approach for Enabling DCT/IDCT Energy Reduction Scalability in MPEG-2 Video Codecs," in *ICASSP*, 2001.

[19] M. Saadatmand *et al.*, "Design of adaptive security mechanisms for real-time embedded systems," in *ESSOS*, 2012.

[20] J. Lee *et al.*, "Optimizing Throughput of Power- and Thermal-Constrained Multicore Processors Using DVFS and Per-Core Power-Gating," in *DAC*, 2009.

[21] F. Paterna *et al.*, "An Efficient On-line Task Allocation Algorithm for QoS and Energy Efficiency in Multicore Multimedia Platforms," in *DATE*, 2011.