Energy Optimization for Latency- and Quality-Constrained Video Applications

Chaeseok Im and Soonhoi Ha
Seoul National University

Editors' note:
The trade-off between energy consumption and media quality is a fundamental design issue in mobile multimedia. This energy optimization technique based on frame skipping and buffering exploits the characteristics of video applications, particularly their tolerance to variation in latency and video quality to increase slack time and its use in dynamic voltage scaling.

—Radu Marculescu, Carnegie Mellon University; and Petru Eles, Linköping University

ONE OF THE MOST IMPORTANT design issues for mobile multimedia applications such as videophones and video cameras is low energy consumption because of their limited energy budgets or battery capacities. Dynamic voltage scaling (DVS) is a prominent low-power technique for such microprocessor-based systems.[1] It reduces a system's total energy consumption by lowering a variable-voltage processor's supply voltage or frequency if slack time remains after the current task execution. The key concern in DVS is to increase slack time use to lower the supply voltage as much as possible under the given performance constraints.

In video applications, it's important and challenging to efficiently provide sufficient quality of service (QoS) in addition to saving energy consumption. Video applications usually have three main characteristics: repetitive processing of streaming input data with varying execution times, tolerance to latency increase, and tolerance to video quality degradation that goes unnoticed by the human eye. We propose a technique that exploits these characteristics, particularly the tolerance to latency, to increase slack time and its use, thus minimizing system energy consumption. We use an H.263 encoder application as a test vehicle for the proposed technique.

The technique has two parts. First, we vary the H.263 encoding algorithm's frame rate according to video quality during the encoding process under a given video quality constraint. Temporal video quality (motion smoothness) depends on the video sequence's frame rate and motion activity. If the frame rate is too low, motion jerkiness annoys the viewer. The obvious way to avoid this is to give video applications a frame rate high enough to accommodate the most motion activity. But such a frame rate would be excessive for scenes with little motion activity. Therefore, our technique adapts the frame rate to the degree of motion activity by skipping redundant frames. Thus, we intentionally create slack time as long as video quality is not unacceptably degraded. We implemented this technique with only a slight modification of the original encoder algorithm and no modification to the decoder.

The second part of the technique is to delay processing within the latency constraint by buffering the input data samples.[2] To guarantee constant throughput, a system design must accommodate the worst-case execution time (*WCET*). When the actual execution time is less than *WCET*, variation in execution time results in workload-variation slack time. However, a system cannot fully use workload-variation slack time when there is no task ready to run in a DVS technique. Our input-buffering technique maximizes the number of pending input samples and thus the number of tasks ready when workload-variation slack time occurs. Designers often use input buffering to

Related work

There has been extensive research on energy optimization techniques based on dynamic voltage scaling. DVS techniques for either static or dynamic real-time applications are usually designed to meet tight application deadlines.[1,2] However, these techniques have paid little attention to quality-of-service (QoS) parameters such as latency and quality. The limitation of previous DVS approaches is that they cannot fully use workload-variation slack time because they cannot avoid idle intervals when no task is ready to run.

For multimedia applications, there is more opportunity for optimization if quality constraints are taken into account. Recently, researchers have introduced energy optimization techniques considering quality constraints. For example, Taylor, Dey, and Panigrahi present an energy optimization technique with latency and image quality constraints for the JPEG image compression algorithm.[3] They construct a table showing the relationship between some of the JPEG algorithm parameters and QoS metrics such as latency and image quality and choose the optimization parameters at runtime for optimal trade-offs between energy and QoS metrics. Cao and Yasuura present an energy and area minimization technique that changes the computation precision of the inverse discrete cosine transform (IDCT)—the kernel program of the MPEG-2 video decoder—under the video quality constraint in terms of both a subjective and an object metrics.[4] Our approach differs from theirs in that we do not modify the algorithm parameters but vary the frame rates by skipping frames under the video quality constraint.

Frame skipping (dropping) is a well-known technique for adapting video to available bandwidth.[5] Song, Kim, and Kuo control the frame rate adaptively according to motion information in a sliding window to reduce video quality variation between adjacent frames.[6] However, their objective is not to reduce energy consumption. Hua, Qu, and Bhattacharyya exploit a given frame-dropping ratio to reduce processor speed and energy consumption.[7] However, they regard the frame-dropping ratio as the QoS metric, ignoring the video quality metric.

References

1. P. Pillai and K.G. Shin, "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems," *Proc. 18th ACM Symp. Operating Systems Principles*, ACM Press, 2001, pp. 89-102.
2. H. Aydin et al., "Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems," *Proc. 22nd IEEE Real-Time Systems Symp.* (RTSS 01), IEEE Press, 2001, pp. 95-105.
3. C.N. Taylor, S. Dey, and D. Panigrahi, "Energy/Latency/Image Quality Tradeoffs in Enabling Mobile Multimedia Communication," *Software Radio—Technologies and Services*, Enrico Del Re, ed., Springer-Verlag, 2000, pp. 55-66.
4. Y. Cao and H. Yasuura, "Quality-Driven Design by Bitwidth Optimization for Video Applications," *Proc. Asia and South Pacific Design Automation Conf.* (ASP-DAC 03), ACM Press, 2003, pp. 532-537.
5. H. Cha, J. Oh, and R. Ha, "Dynamic Frame Dropping for Bandwidth Control in MPEG Streaming System," *Multimedia Tools and Applications*, vol. 19, no. 2, Feb. 2003, pp. 155-178.
6. H. Song, J. Kim, and C.-C.J. Kuo, "Real-Time Encoding Frame Rate Control for H.263+ Video over the Internet," *Signal Processing: Image Communication*, vol. 15, nos. 1-2, Sept. 1999, pp. 127-148.
7. S. Hua, G. Qu, and S.S. Bhattacharyya, "Energy Reduction Techniques for Multimedia Applications with Tolerance to Deadline Misses," *Proc. 40th Design Automation Conf.* (DAC 03), ACM Press, 2003, pp. 131-136.

stabilize fluctuating interarrival times of input frames by buffering several frames beforehand for a constant input consumption rate. Therefore, our buffering technique does not incur significant resource overhead. (The "Related work" sidebar describes other DVS-based energy optimization techniques.)

Buffering technique

Our buffering technique maximizes slack time use under a latency constraint. Consider a periodic task in which *WCET* equals the period and the actual execution time is half of *WCET*. A typical intertask DVS technique without input buffering cannot exploit slack time fully because a new task instance can't be scheduled before the task's release point. Thus, as Figure 1a shows, the processor is in idle state half the time. On the other hand, we can schedule the next task instance to exploit slack time fully by buffering one input packet beforehand, as Figure 1b shows. Even though the processor in idle state can go into power-down mode in the former case, there is more energy saving in the latter case. As the arrows in the figure indicate, buffering increases latency. So we

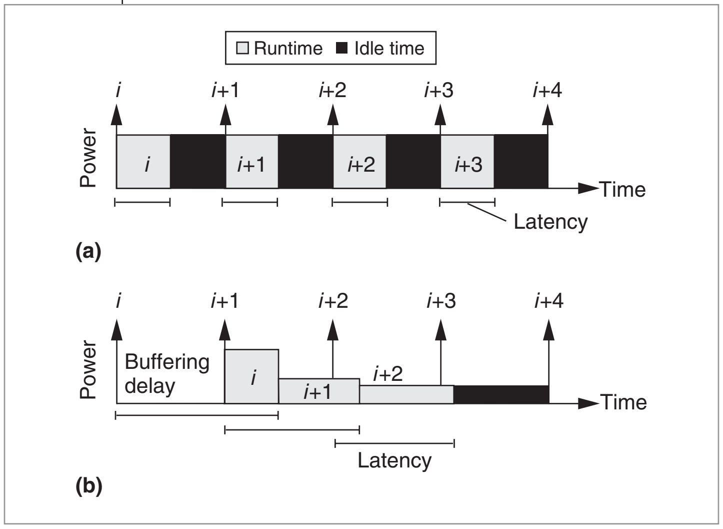


Figure 1. Dynamic voltage scaling without buffering (a) and with buffering (b).

can use the buffering technique as long as buffering delay is acceptable within the latency constraint.

The minimum buffer size to achieve maximum energy saving is

$$\min\left(\left\lceil \frac{WCET}{BCET} - 1 \right\rceil, \left\lfloor \frac{T_{LC}}{T_P} - 1 \right\rfloor\right)$$

for the single-task situation, where *WCET* and *BCET* are the task's worst-case and best-case execution times, and T_{LC} and T_P are the latency constraint and the task period.[2]

By enhancing the earliest-deadline-first (EDF) and the rate-monotonic (RM) algorithms with the buffering technique, we can easily extend the technique to multitask cases.[3]

Motivation for frame skipping

For quantitative analysis, we use motion smoothness as an objective measure of video quality. If we define frame difference *FD* as the mean-square difference between two adjacent frames, we can quantify video quality using the following formula:

$$Quality = 10\log_{10}\left[\frac{1}{FD}(255^2)\right]\left[dB\right]$$

This equation, which looks like the measurement for peak signal-to-noise ratio (PSNR), calculates quality for an 8-bit image.

Figure 2a shows the average video quality for several video sequences at different frame encoding rates. Generally, a video application operates at a fixed frame rate—as high as 30 frames per second (fps) in our example—to satisfy the video quality requirement even at worst-case scene changes. In this experiment, we reduced the frame rate by skipping the frames periodically. To achieve 7.5 (or 30/4) fps, for example, we skip the next three frames after encoding one frame. Video quality decreases linearly as the frame rate decreases for all video sequences. However, video quality also depends on the video sequences' motion activity. Because motion activity is different among sequences, so are video quality and its variations among frame rates:

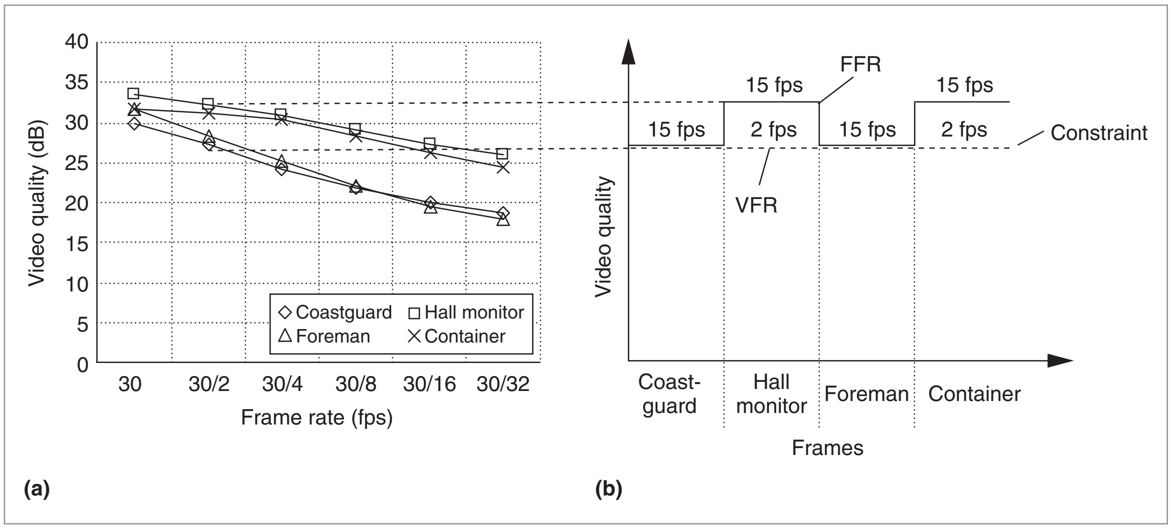


Figure 2. Video quality comparison according to frame rate control schemes: Average video quality vs. encoding frame rate (a); fixed-frame-rate (FFR) scheme vs. variable-frame-rate (VFR) scheme (b).

The Coastguard and Foreman video sequences have more-dynamic scenes than the other two examples, so they show lower video quality for all frame rates. On the other hand, Hall Monitor and Container show higher video quality because of their low motion activity.

Therefore, as Figure 2b shows, if a video sequence composed of variant video scenes is encoded with a fixed frame rate (FFR), its video quality will fluctuate, having far higher quality than required by the video quality constraint for a significant portion of time.

The basic idea of our technique is to skip frames in the encoding process if the video quality is higher than a given constraint. The frame-skipping technique with the help of the DVS technique decreases the encoding task's effective frame rate to increase slack time and reduce energy consumption. We use a variable frame rate (VFR), skipping more frames during Hall Monitor and Container, to more uniformly distribute the video quality above the constraint, as Figure 2b shows.

Frame-skipping techniques

We use two frame-skipping techniques: direct comparison and forward prediction. We use the first technique when we want to skip frames while keeping the video quality constraint. We use the second when we want to skip frames without adding buffer and latency overhead.

Direct comparison

Figure 3 illustrates the direct-comparison frame-skipping technique. Conceptually, the simple and intuitive technique includes two concurrent processes: frame skipping and encoding. Frame skipping selects the next frame to be encoded by comparing the currently selected image with newly captured images in turn until the difference in temporal video quality between two images violates the given quality constraint or until the latency constraint is violated. If a violation occurs, this process selects the previously captured image as the frame to be encoded. The encoding process encodes only the frames selected by the frame-skipping process. The interval between two consecutively encoded frames, called the frame interval, varies dynamically and ranges from 1 to the maximum frame interval (*MFI*), which is limited by the given latency constraint.

Although the direct-comparison technique naturally requires look-ahead buffers, it does not violate the given video quality constraint because it performs the validation process beforehand. Therefore, it combines seamlessly with the buffering technique. The maximum buffer requirement equals *MFI*, and the additional buffers increase latency. The maximum latency becomes $2(MFI)T_P$: $MFI(T_P)$ for the initial buffering, and another $MFI(T_P)$ for the encoding itself with the buffering technique. Therefore, *MFI* must satisfy the following inequality:

$$MFI \leq \frac{T_{LC}}{2T_P}$$

Figure 4 presents the algorithm for the direct-comparison frame-skipping process. Its total overhead, including the PSNR calculation, voltage switching, and thread switching, is less than 5% of the optimized H.263 encoder's overhead.

Figure 5 shows an example scenario. Suppose that *MFI* is 4. At first, the frame-skipping process buffers the initial four frames and selects frames 0 and 1 to be encoded. Then the encoding process encodes these frames by using DVS during their frame interval. At the same time, the frame-skipping process continues to receive the subsequent frame and selects frames 3, 6, and 8 for encoding. Figure 5a shows the supply voltage variation, assuming the encoding task always takes its *WCET*.

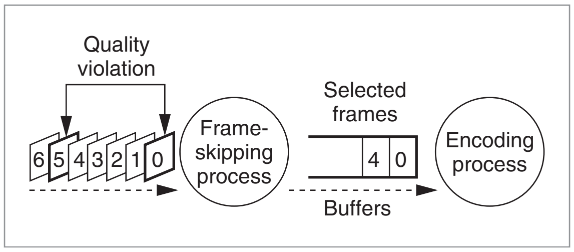
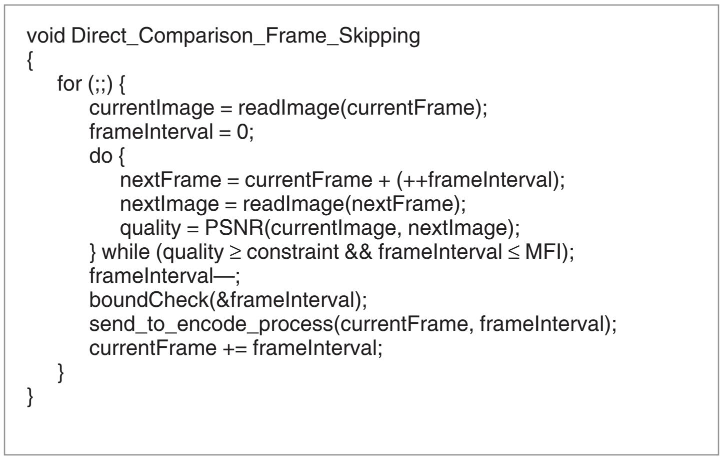


Figure 3. Direct-comparison frame-skipping technique.



```
void Direct_Comparison_Frame_Skipping
{
    for (;;) {
        currentImage = readImage(currentFrame);
        frameInterval = 0;
        do {
            nextFrame = currentFrame + (++frameInterval);
            nextImage = readImage(nextFrame);
            quality = PSNR(currentImage, nextImage);
        } while (quality ≥ constraint && frameInterval ≤ MFI);
        frameInterval—;
        boundCheck(&frameInterval);
        send_to_encode_process(currentFrame, frameInterval);
        currentFrame += frameInterval;
    }
}
```

Figure 4. Direct-comparison frame-skipping algorithm.

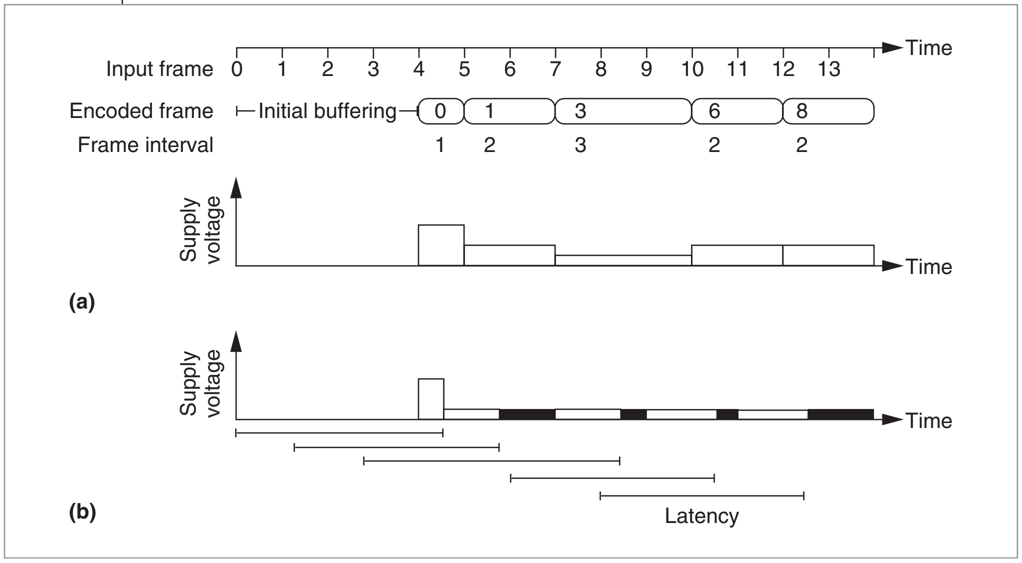


Figure 5. Direct-comparison-technique encoding example (buffer size = 4): when the encoding task always takes its *WCET* (a); when actual encoding time (*AET*) is half of *WCET* (b).

```
void Prediction_Frame_Skipping
{
    frameInterval = fixedFrameInterval;
    for (;;) {
        currentImage = readFrame(currentFrame);
        quality = PSNR(currentImage, previouslyReconstructedImage);
        if (quality > constraint + α)
            frameInterval++;
        else if (constraint ≤ quality ≤ constraint + α)
            ; // preserve the current frame interval
        else if (constraint – α ≤ quality < constraint)
            frameInterval /= 2;
        else if (quality < constraint – α)
            frameInterval = fixedFrameInterval;
        boundCheck(&frameInterval);
        DVS(frameInterval);
        doEncoding(currentFrame);
        skipFrames(frameInterval – 1);
    }
}
```

Figure 6. Forward-prediction algorithm.

If we assume that the actual encoding time (*AET*) is half of *WCET*, the CPU becomes idle during half the interval, so we can further reduce the supply voltage by using workload-variation slack time, as Figure 5b shows. In the direct technique, a frame can be encoded only after the next frame to encode is selected. Therefore, frames 1, 3, 6, and 8 are ready for encoding at times 4, 7, 9, and 11, respectively.

Forward prediction

Whereas the direct technique delays the current frame's encoding until the next frame is selected, the forward-prediction technique predicts the frame interval based on the history. Initially, we set the frame interval to 1, meaning no frame is skipped. Upon the next frame's arrival, we compute the difference between the frame and the previously reconstructed image. If the difference is less than the quality constraint, we increase the frame-skipping interval and start the encoding with the adjusted voltage and frequency, using the full frame interval. Because latency and buffer overhead do not exist in the same capturing stage, the *MFI* increase is double that of the direct technique. Thus, the following inequality holds:

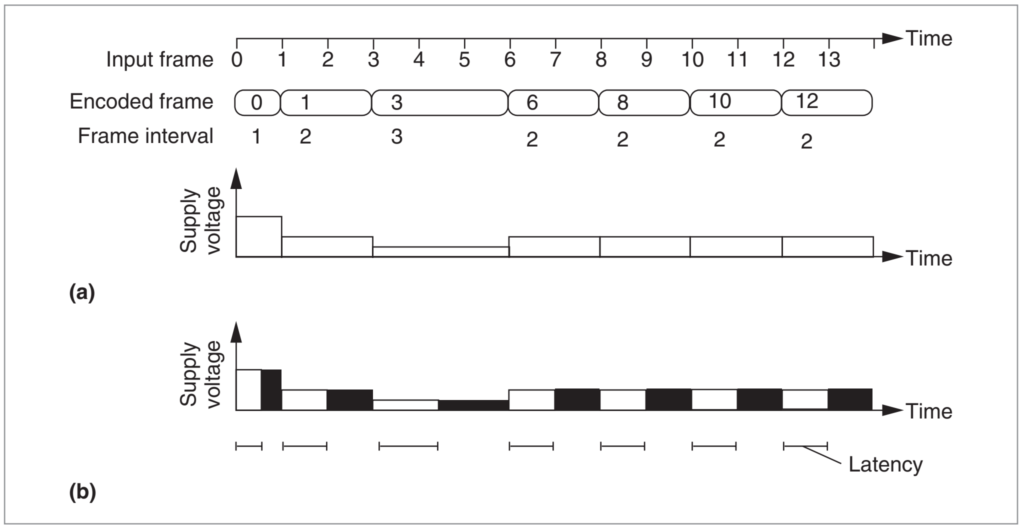
$$MFI \leq \frac{T_{LC}}{T_P}$$

If the latency constraint is tight, the prediction technique is preferable to the direct technique. However, the prediction technique might violate the quality constraint if the prediction fails. Figure 6 shows the prediction algorithm, which resulted in a quality violation rate of less than 1% in our experiments.

After reading the current frame, we calculate the quality between the previously reconstructed and the current frame. Depending on the quality, we adjust the frame interval as follows, using the margin parameter α, which we defined as 2 dB through preliminary experiments:

- Case 1: *quality* > *constraint* + α. This means there is no abrupt scene change during the previous frame interval. Then, we increase the frame interval by 1, assuming this quiescent situation will continue during the next frame interval.
- Case 2: *constraint* ≤ *quality* ≤ *constraint* + α. This region is where we want the application to operate. So, we don't change the frame interval but use the previous frame interval as the current one.
- Case 3: *constraint* – α ≤ *quality* < *constraint*. This means the previous frame interval is too large to

accommodate the scene change. Therefore, we must reduce the frame interval to raise the quality above the constraint. For faster adaptation, we reduce the frame interval by half, or double the frame rate.
- Case 4: *quality* < *constraint* – α: This indicates that the quality became seriously worse during the previous frame interval, a situation that usually occurs with abrupt scene change. We set the frame rate to the maximum to recover the same video quality as that of the default FFR scheme.



Time
Input frame 0 1 2 3 4 5 6 7 8 9 10 11 12 13
Encoded frame 0 1 3 6 8 10 12
Frame interval 1 2 3 2 2 2 2
Supply voltage
Time
(a)
Supply voltage
Time
Latency
(b)

Figure 7. Prediction-technique encoding example: when the encoding task always takes its *WCET* (a); when *AET* is half of *WCET* (b).

In the latter two cases, some previous frames might violate the video quality constraint. The uncertainty of the captured input frames makes this unavoidable. Because the constraint violation percentage depends on the parameter value of α, we set this value conservatively. As the frame interval becomes large, so does the probability that scene change will occur during the next frame interval. Therefore, we set a maximum frame interval even if the quiescent situation lasts longer.

Compared with direct comparison, forward prediction has several benefits:

- It doesn't require a separate frame-skipping process. Instead, it augments the encoding task. Thus, the algorithm's total overhead is smaller.
- Maximum latency is half that of direct comparison.
- It doesn't require an extra buffer.

Figure 7 shows an example of the prediction technique's encoding. It's similar to that of the direct technique except that it doesn't use any buffering for encoding. Therefore, as Figure 7b shows, we cannot use workload-variation slack time if *AET* is less than *WCET*.

Experiments

We used TMN version 3.0 from the University of British Columbia Image Processing Laboratory for the reference H.263 encoder. We used a default encoder setup; the quantization parameter was 13, and the target bit rate was variable. We didn't consider B frames in our experiment.

We assumed that the application runs on an ARM8 processor that operates from 5 MHz with 1.2 V and 2.4 mW to 80 MHz with 3.4 V and 381 mW.[4] For a voltage change, the transition time is 520 μs, and the transition energy is 130 μJ. These time and energy overheads are negligible considering that the frame-encoding period is 33 ms, and the period's energy consumption is 12.6 mJ (381 mW × 33 ms) in 30 fps. We obtained actual execution cycles from an ARM processor simulator, ARMulator. We determined the operational frequency dynamically by dividing the maximum frequency by the frame interval. From an ARM8 processor power-frequency table, we obtained the power dissipation corresponding to a given frequency and calculated each frame's energy consumption and the total energy consumption.

To mimic real scene variation, we composed four 300-frame common intermediate format (CIF) video sequences—Coastguard, Hall Monitor, Foreman, and Container—into a single 1,200-frame sequence. Coastguard and Foreman have more motion than the others. We set the maximum frame rate to 30 fps for all of them.

Video quality distribution

Figure 8 illustrates how our technique operates with a given video quality constraint. In Figure 8a, we plot video quality distribution of the FFR scheme, and in Figure 8b, the VFR schemes with the video quality constraint of 25 dB. Figure 8b distinguishes two VFR schemes: direct and prediction. Under the FFR scheme, video quality fluctu-

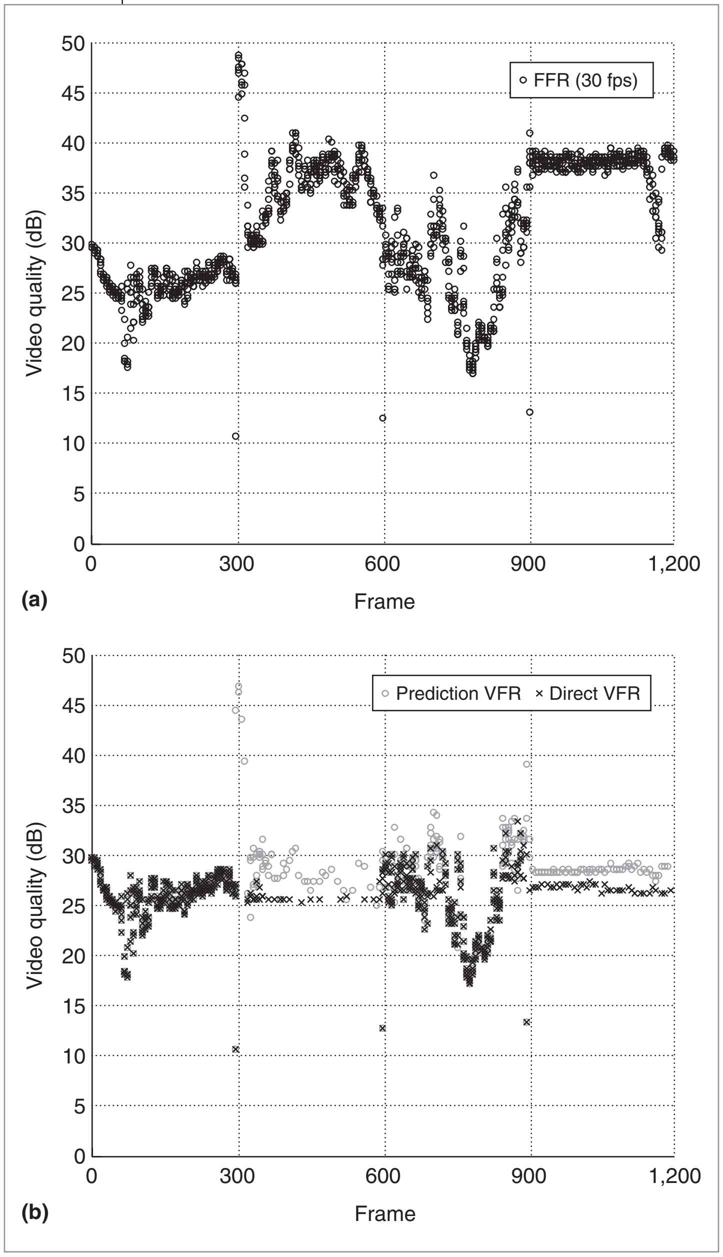


Figure 8. Comparison of video quality distribution: FFR scheme at 30 fps (a); VFR schemes with a constraint of 25 dB (b).

ates according to the original video sequence's motion activity; under the VFR schemes, video quality is stable. Even at the maximum frame rate of 30 fps, some frames have lower video quality than the constraint requires.

The direct scheme guarantees that all frames meet the quality constraint, whereas the prediction scheme results in some frames that violate the constraint. The violation rate in these experiments was less than 1%. Another difference is that the direct scheme maintains fairly stable video quality independently of scene variations, whereas the prediction scheme shows wider quality variance caused by online adaptation of frame intervals.

Performance with video quality constraints

Now we show performance comparisons between diverse energy optimization approaches using various video quality constraints without the latency constraint. We compare six schemes:

- FFR scheme with power-down only,
- FFR scheme with buffering technique,
- direct VFR scheme with power-down only,
- direct VFR scheme with buffering technique,
- prediction VFR scheme with power-down only, and
- prediction VFR scheme with DVS.

We varied the video quality constraint as 15, 20, 25, 30, and 35 dB for the VFR schemes. Figure 9a shows the average frame rates for given video quality constraints. At the video quality constraint of 35 dB, all average frame rates are around 30 fps, but at the constraint of 15 dB, they decrease to 2.25 fps with the direct scheme and 3.5 fps with the prediction scheme. The prediction scheme skips fewer frames than the direct scheme because of a conservative setting of margin parameter α. The average frame rates decrease linearly with the video quality constraint.

Figure 9b compares energy consumption for given video quality constraints. Energy consumption is normalized to that of the FFR scheme with power-down only. The figure shows that the buffering-alone scheme (FFR with buffering) reduces energy consumption by 55%. In comparison, the frame-skipping-alone scheme (prediction VFR) shows better performance when the given video quality constraint is low enough to reduce the frame rate sufficiently. The direct scheme shows better performance than the prediction scheme (if the latency constraint is not considered) because the former has fewer frames than the latter. We saved the most energy by applying both the buffering and the frame-skipping techniques together (direct VFR with buffering).

Figure 10 shows the direct frame-skipping technique's bit rate change with given video quality constraints. The encoded bits per frame increase as more frames are skipped. The reason is that motion estimation fails as more macroblocks are intracoded because of the increased prediction error. However, the reduced frame rate makes the overall bit rate (bps) decrease as the quality constraint decreases.

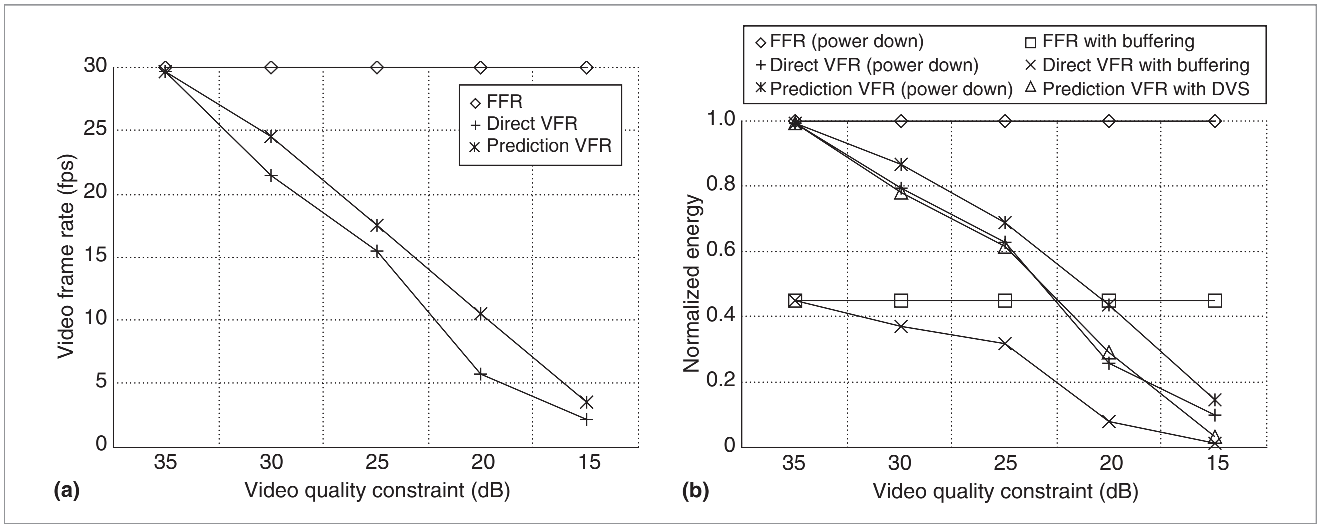


Figure 9. Performance of various schemes with video quality constraints: average frame rate (a), and normalized energy consumption (b).

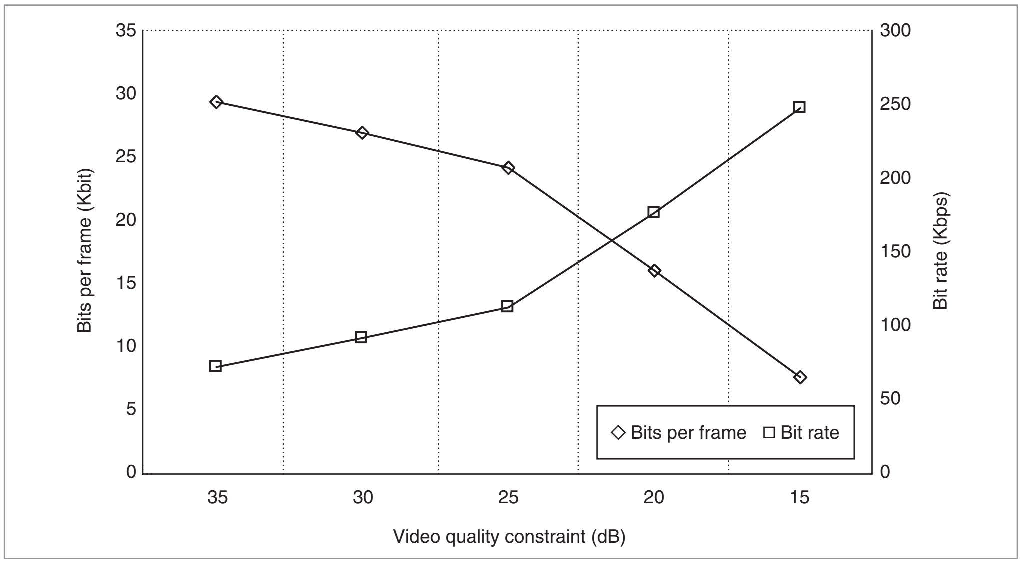


Figure 10. Direct frame-skipping technique: bit rate vs. video quality constraint.

Performance with latency constraints

Figure 11 shows the various schemes' average frame rates and energy consumption changes with latency constraints, where T is the period of the task. The video quality constraint is 20 dB, and the results in Figure 9 serve as the base performance of the infinite latency constraint.

In Figure 11a, the prediction scheme shows a lower average frame rate than the direct scheme when the latency constraint is tight. The reason is that at the same latency constraint, the prediction scheme's *MFI* is larger than the direct scheme's *MFI*, so the former scheme can skip more frames than the latter. The direct scheme does not decrease the frame rate below the latency constraint of $4T$ because the direct scheme needs at least two additional buffers for frame skipping, and they incur the initial latency of at least $4T$. Also, there is no change in $3T$ and $5T$ because an additional buffer increases latency by $2T$ in the direct scheme.

The energy consumption results show a similar pattern in Figure 11b. All the proposed schemes, except the direct scheme, achieve noticeable performance enhancement, even at the $2T$ latency constraint. On the other hand, the direct scheme shows no performance improvement at latency constraints below $4T$. The direct scheme's performance gets better as the latency constraint gets looser.

Our future work will aim to make the proposed video quality measure more sensitive to local motion by partitioning the whole picture into subpictures and investigating the picture difference between them. We also seek a more effective and less computationally intensive

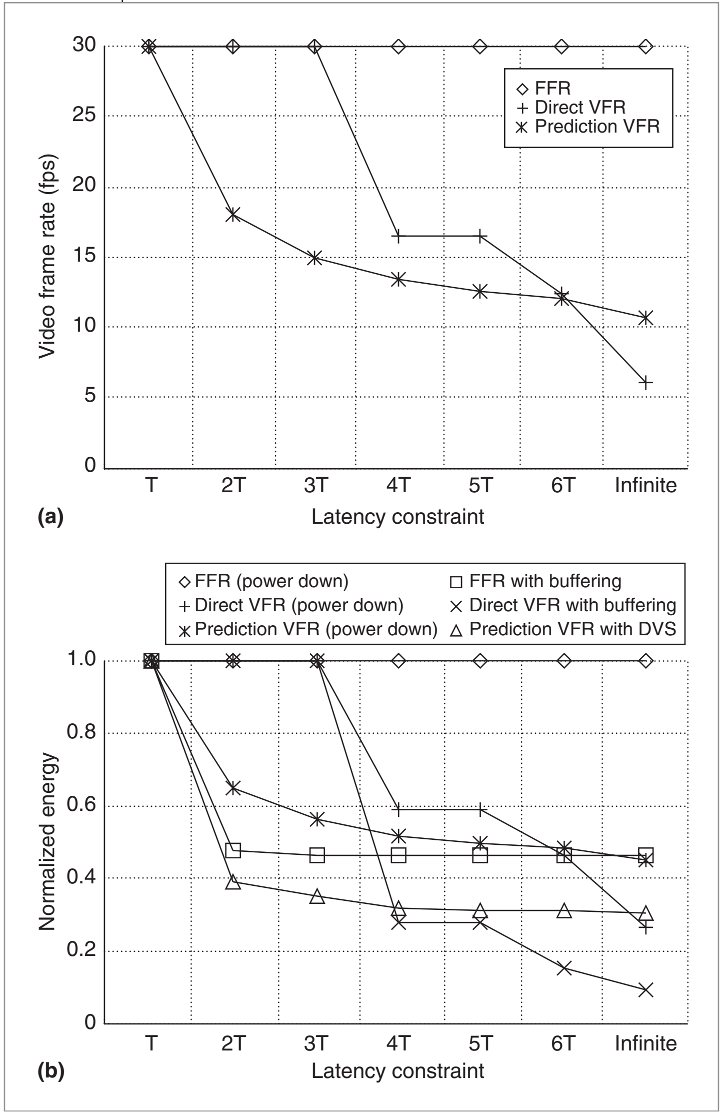


Figure 11. Performance of various schemes with latency constraints: average frame rate (a), and normalized energy consumption (b).

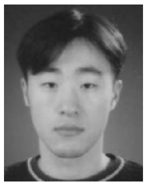
video quality measure. Multitask extension of this work will face challenging issues such as synchronization and data dependency between data streams. Applying this work to cases where transmission errors occur in channels can be another interesting research topic. ■

Acknowledgments

National Research Laboratory Program No. M1-0104-00-0015 and the Brain Korea 21 Project supported this work. The Institute of Computer Technology at Seoul National University provided research facilities.

References

1. A.P. Chandrakasan, S. Sheng, and R.W. Brodersen, "Low-Power CMOS Digital Design," *IEEE J. Solid-State Circuits,* vol. 27, no. 4, Apr. 1992, pp. 473-484.
2. C. Im and S. Ha, "Dynamic Voltage Scheduling for Low-Power Multimedia Applications Using Buffers," *Proc. Int'l Symp. Low Power Electronics and Design* (ISLPED 01), IEEE Press, 2001, pp. 34-39.
3. C. Im and S. Ha, "Dynamic Voltage Scaling for Real-Time Multi-Task Scheduling Using Buffers," *Proc. ACM Conf. Languages, Compilers, and Tools for Embedded Systems* (LCTES 04), ACM Press, 2004, pp. 88-94.
4. T.D. Burd and R.W. Brodersen, "Design Issues for Dynamic Voltage Scaling," *Proc. Int'l Symp. Low Power Electronics and Design* (ISLPED 00), IEEE Press, 2000, pp. 9-14.



Chaeseok Im is a PhD candidate in the School of Electrical Engineering and Computer Science of Seoul National University, Korea. His research interests include low-power multimedia system design, system-level energy estimation, and low-power embedded operating systems. Im has a BS and an MS, both in computer engineering, from Seoul National University. He is a student member of the ACM.

Soonhoi Ha is an associate professor in the School of Computer Science and Engineering at Seoul National University. His research interests include hardware-software codesign and design methodology for embedded systems. Ha has a BS and an MS in electronics engineering from Seoul National University and a PhD in electrical engineering and computer science from the University of California, Berkeley. He is a member of the ACM and the IEEE Computer Society.

■ Direct questions and comments about this article to Chaeseok Im, Seoul National University, School of Computer Science and Eng., Rm. 456, Bldg. 301, Shinlim-dong, Gwanak-gu, Seoul 151-744 Korea; csim@iris.snu.ac.kr.