Application-specific Workload Shaping in Multimedia-enabled Personal Mobile Devices

Balaji Raman Samarjit Chakraborty
Department of Computer Science, National University of Singapore
E-mail: {ramanbal,samarjit}@comp.nus.edu.sg

ABSTRACT

Today, most personal mobile devices (e.g. cell phones and PDAs) are multimedia-enabled and support a variety of concurrently running applications such as audio/video players, word processors and web browsers. Media-processing applications are often computationally expensive and most of these devices typically have 100 – 400 MHz processors. As a result, the user-perceived application response times are often poor when multiple applications are concurrently fired. In this paper we show that by using application-specific dynamic buffering techniques, the workload of these applications can be suitably "shaped" to fit the available processor bandwidth. Our techniques are analogous to *traffic shaping* which is widely used in communication networks to optimally utilize network bandwidth. Such shaping techniques have recently attracted a lot of attention in the context of embedded systems design (e.g. for dynamic voltage scaling). However, they have not been exploited for enhanced schedulability of multiple applications, as we do in this paper.

Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: Special-purpose and application-based systems—*Real-time and embedded systems*

General Terms

Algorithms, Performance, Design

Keywords

Multimedia systems, schedulability analysis, mobile devices

1. INTRODUCTION

The last few years have seen a huge proliferation of personal mobile devices such as PDAs, cell phones, portable audio/video players and gaming devices. Many of these devices now support multiple functionalities, have an operating system running on them and allow multiple applications to be run concurrently. For example, a common use scenario for a PDA is to play an audio or a video clip, and at the same time use a *Datebook* application. These applications often exhibit poor response times when run on the 100 – 400 MHz processors found in most personal mobile devices.

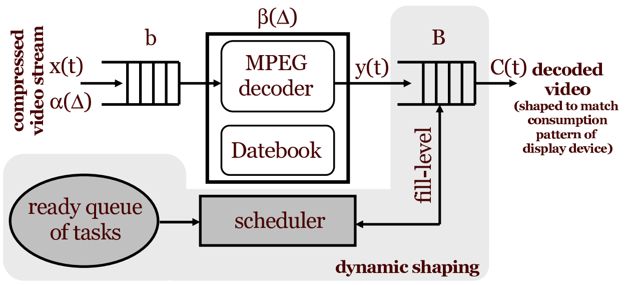

Figure 1: Setup for dynamic workload shaping.

Clearly, this problem will become more acute as the user demand increases in terms of the number of applications being concurrently run, they becoming increasingly rich in graphics/animation and the increasing use of audio/video content in mobile devices.

Although a lot of existing work from the processor scheduling domain (especially in the context of multimedia applications) addresses this problem, in this paper we look at it from a different perspective. We show that by using *dynamic buffering* techniques, the workload of different concurrently running applications can be appropriately *shaped* to fit the available processor bandwidth. Although buffering is a well-established technique to smooth out the variabilities associated with continuous media streams, it has predominantly been used in the context of streaming media (i.e. when the audio/video data is downloaded from the network before being processed). To ensure continuous playout, the buffering time or playout delay in such cases is determined based on the network conditions.

However, much less importance is attached to buffering in the case of stored media. Here, the playout delay is typically very small and is chosen in an ad hoc fashion, without taking into account the characteristics of the application or the media stream being processed. More importantly, the playout delay is chosen independently of the other tasks running on the processor. Further, current processor scheduling techniques do not exploit buffering as a means for shaping the workload associated with the tasks being scheduled. We show that by appropriately addressing these three issues, the schedulability of multiple concurrently running tasks can be substantially enhanced.

Our contribution: Towards this, we present a mathematical model that can be used to analyze applications processing continuous media streams, and setups where such applications concurrently run with other applications processing more static input data. More specifically, we show that the choice of the playout delay has a significant impact on the workload generated by such applications. Given the available processor bandwidth, our model can be used to calculate the minimum playout delay using which an application may be supported by this available bandwidth. We also show that the buffer fill level can be dynamically changed at runtime to periodically free up sufficient processor bandwidth to support other concurrently running applications. The parameters of such dynamic scheduling and buffer management policies are application specific and can be calculated using our model. Determining these parameters by trial and error and repeatedly validating them using simulation is expensive in terms of the simulation time involved and is also highly error prone. On the other hand, our model requires each application to be simulated only once, and in isolation, using representative input data (or audio/video clips) to determine the values of certain parameters characterizing the application. These parameters then serve as input to our mathematical model and are used to determine scheduling and buffer management policies when multiple applications run concurrently. Simulating the execution of each application in isolation is considerably easier than a full system simulation where multiple applications run together and get preempted by the scheduler.

Relation to previous work: The basic idea we exploit in this paper is similar to *traffic shaping*, which is a well-established technique in the communication networks domain [1, 5, 6]. A traffic shaper is used to buffer network packets from an incoming packet stream and delay them so that the outgoing stream from the shaper conforms to a pre-defined traffic specification. Such shaping is used to smooth out the burstiness in the packet stream, thereby preventing such burstiness to accumulate as the stream passes from one network node to the next. This results in improved network bandwidth utilization and reduces global buffer requirements. We, on the other hand, exploit buffering to smooth out the variabilities in the *execution requirements* of media processing applications. The aim is to shape the workload arising from such applications, so that it can be served at an "average" rate by appropriately choosing the initial playout delay or buffering time. Further, we dynamically change the amount of buffering associated with a running application to free up sufficient processor bandwidth in response to new tasks fired by an user. The novelty of our work stems from the analogy we establish between our representation of the variability in the execution requirements of an application, and existing models for quantifying burstiness in network traffic.

Recently shaping techniques have also attracted a lot of interest within the embedded systems domain. Shaping has been used as a means for aggregating the workload of media-processing applications (e.g. video decoders) to create idle periods. Such idle periods are then exploited to shut-off a processor or scale down its operating voltage/frequency to save power (see [2, 3, 9, 11] for applications of this scheme in different setups). Very recently, shaping has also been used in the particular context of designing multi-processor System-on-Chip (SoC) architectures. It has been shown that shaping on-chip traffic leads to reduced on-chip global buffer requirements and improves overall system performance and predictability [8, 14]. Similar techniques have also been shown to be useful when applied to Networks-on-Chip (NoC) architectures, where again they result in improved worst-case response times and global buffer requirements (which in turn lead to reduced chip area and power consumption) [10]. Our work in this paper follows this line of research and specifically shows that application-specific buffering can be used to shape the workload of real-time applications processing continuous media streams, and such shaping can be exploited to significantly improve the overall schedulability of a pre-defined set of applications.

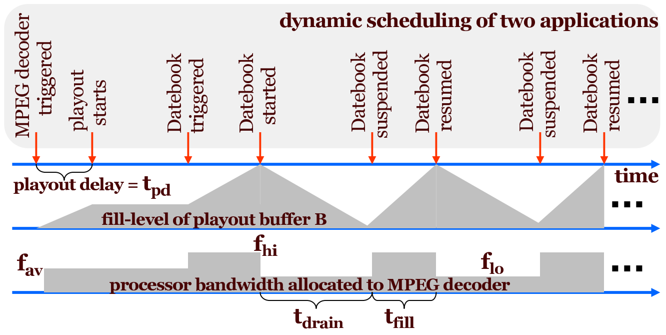

Figure 2: Dynamically controlling the playout buffer fill level as two applications are being scheduled.

Organization of the paper: In the next section we present an example to illustrate how the amount of buffering associated with a running application is dynamically changed as new tasks are triggered for execution. More specifically, we use this example to identify the scheduling/buffer management *parameters* associated with each application. In Sections 3 and 4 we then show how the values of these parameters can be estimated from our model for characterization the execution demand of an application on a specified processor architecture. Further, Section 3 shows that the workload of an application is heavily influenced by the initial playout delay or buffering time associated with it. The model proposed in this section is used to quantitatively capture this tradeoff. This model is then used in Section 4 to describe our dynamic buffering policy. Finally, our experimental results in Section 5 show how the proposed schemes can be used to improve the schedulability of a pre-defined set of tasks consisting of media-processing and interactive text-processing (e.g. *Datebook*) applications.

2. ILLUSTRATIVE EXAMPLE

Figure 1 is a high-level view of our setup. It shows a processor running two tasks, an MPEG-2 decoder and a Datebook application that is commonly supported on PDAs. The input to the decoder is a compressed video stream that arrives at the input buffer b. This buffer is read by the decoder and the decoded video stream is written into the playout buffer B, which in turn is read by the playout (or display) device at a pre-defined rate (e.g. 25 frames/sec). Throughout this paper we assume that the tasks running on the processor are scheduled using some *proportional-share* scheduler [4, 7] which allocates a specified amount of CPU bandwidth to each task. Our goal in this paper is to show how such a scheduler[1] can exploit dynamic buffering to enhance the schedulability of a set of applications. Towards this, we modify the scheduler to dynamically change the processor share allocated to each task at runtime. These shares, as we show in this paper, are determined by the fill-level of the playout buffer B, the execution demands of the running tasks (which we characterize using a mathematical model in Section 3), and the set of tasks in the ready-queue of the scheduler (see Figure 1).

[1]Our scheme can be applied to other scheduling disciplines as well. But for simplicity of exposition, we only restrict ourselves to proportional-share schedulers in this paper.

To illustrate our scheme, let us consider the scenario shown in Figure 2. We assume that our processor has an effective bandwidth of f_{max} MHz available for running user applications (i.e. after supporting operating system tasks). At time $t = 0$, the user triggers the MPEG decoder application, which is started immediately. However, the playout from the buffer B only starts at $t = t_{pd}$, which is the playout delay. With t_{pd} as the playout delay, the decoder occupies a processor bandwidth of f_{av} MHz. At time $t = t'$ the user now triggers the Datebook application which requires a processor bandwidth of f_{db}. However, it turns out that $f_{max} - f_{av} < f_{db}$ and hence the Datebook task cannot be executed (immediately). In response to this, the scheduler increases the processor bandwidth allocated to the decoder task, from f_{av} to f_{hi} (where $f_{hi} \leq f_{max}$). With this bandwidth, the average decoding rate is higher than the consumption rate of the output device from the playout buffer B. This results in the fill-level of B to increase till the time $t = t' + t_{fill}$, when the allocated bandwidth to the decoder is reduced to f_{lo} ($< f_{av}$). f_{lo} is chosen such that the freed bandwidth (i.e. $f_{max} - f_{lo}$) sufficient to support the Datebook task. Hence, this task starts at $t = t' + t_{fill}$ and continues till $t = t' + t_{fill} + t_{drain}$. During the time interval $[t' + t_{fill},\ t' + t_{fill} + t_{drain})$ the fill-level of B continuously decreases because the bandwidth f_{lo} is not sufficient to sustain the playout rate demanded by the output device. $t' + t_{fill} + t_{drain}$ is the earliest time[2] at which B is fully drained. At this time, the processor bandwidth allocated to the decoder is again increased to f_{hi} for the next t_{fill} time units and this cycle is repeated till the Datebook task is terminated by the user.

Note that this scheme will work if t_{fill} is relatively small compared to t_{drain}. For many applications such as Datebook, which involves interactive text processing and input from an user, this is indeed the case and the (small) periodic time intervals during which the task is suspended are tolerable.

Schedulability analysis: To formally analyze this setup, we model the Datebook application as a periodic task, with an execution requirement, period and deadline. Such a model is general enough to capture a wide variety of applications. The decoder application, on the other hand, processes a continuous media stream and is modelled differently. The performance constraint that needs to be satisfied in this case is that the output device should always be able to read a decoded video frame from the playout buffer B. Given the playout rate of the output device, this translates to the constraint that the playout buffer should never *underflow*.

Hence, our problem reduces to a schedulability analysis problem where a system designer has to estimate whether the media-processing task can satisfy its buffer underflow constraint and the periodic task its deadline constraint. However, in contrast to classical schedulability analysis problems, here it leads to the following question: Given the execution demands of the two tasks (which we formally model in Section 3) does there exist t_{fill}, t_{drain} and f_{lo}, such that the buffer underflow and the deadline constraints are satisfied? For setups where the answer to this question is "*yes*", the designer would additionally want to know *all* possible values of t_{fill}, t_{drain} and f_{lo} which lead to a schedulable system. In the following sections we show how to address this problem.

In summary, we would like to emphasize that our scheme attempts to shape (lower-bound) the output from the playout buffer B to closely match the consumption pattern of the output (display) device. This in turn shapes the workload of the media-processing application to create slacks which are used to accommodate a periodic task.

[2]The rate at which the decoded video is written into B is variable, because of the data-dependent variability in the decoding time of each video frame/macroblock. Hence, it might take longer for B to become empty.

3. BUFFERING TIME VERSUS WORKLOAD

In this section we present a model to characterize the workload of continuous media-processing applications (e.g. the MPEG decoder described in Section 2). Our main result is that the processor bandwidth required to sustain a specified playout rate depends heavily on the initial playout delay t_{pd}. We show that our model can be used to quantitatively characterize this tradeoff. Our dynamic buffering scheme exploits this observation and in Section 4 our model is used to develop the schedulability test we outlined in Section 2.

Before presenting the model, we would first like to explain the intuition behind our scheme. Most multimedia applications exhibit data-dependent variability in their execution requirements. In other words, when such an application processes a stream of data items (e.g. macroblocks or frames in the case of MPEG decoding), the number of processor cycles required to process each data item is highly variable. The ratio of the worst-case and the average load on a processor running such an application can be as high as a factor of 10 [12]. The playout rate associated with the application imposes certain real-time constraints on it. When the playout delay is negligible, such constraints translate to an upper bound on the time that can be spent in processing each data item. Since the number of processor cycles required by each data item is variable, the minimum processor bandwidth is determined by the item which requires the maximum number of processor cycles. When the playout of the application is delayed (i.e. the processed data items are buffered before being played out), the minimum required processor bandwidth decreases. For any given delay, the "amount" of decrease is proportional to the variability in the execution requirement of the stream. With a sufficiently large playout delay, the minimum required processor bandwidth corresponds to the average processor cycle requirement per data item.

Further, many multimedia tasks have variable input-output rates, i.e. the number of input data items consumed to produce one processed data item at the output is variable [13]. For example, the variable length decoding task in an MPEG decoder consumes a variable number of bits to produce one partially decoded macroblock. This provides additional possibility for reducing the required processor frequency by buffering the decoded frames before playout.

3.1 System Model

In what follows, we show how to precisely characterize these variabilities in the execution demand and input/output rates and how this translates into quantifying the tradeoff between processor bandwidth and playout delay. Once again, we use the setup shown in Figure 1.

For the media-processing application (MPEG decoder) We assume that the input bit stream to be processed is fed into the buffer b at a constant rate of r bits/sec. Further, for the sake of generality, we will consider a stream to be made up of a sequence of *stream objects*. A stream object might be a macroblock in the case of video decoding or a granule in the case of audio decoding tasks. Now, given a stream to be decoded, let $x(t)$ denote the number of stream objects arriving at b over the time interval $[0, t]$. Due to the variability in the number of bits constituting a stream object, the function $x(t)$ varies from stream to stream even when their arrival rates are fixed at r bits/sec. We define two functions $\alpha^l(\Delta)$ and $\alpha^u(\Delta)$ to bound the variability in the arrival process of the stream objects at b. These two functions are defined as:

$$\alpha^l(\Delta) \leq x(t+\Delta) - x(t) \leq \alpha^u(\Delta)$$

for all t and $\Delta \geq 0$, where $\alpha^l(\Delta)$ and $\alpha^u(\Delta)$ denote the minimum and maximum number of stream objects that can arrive at b within *any* time interval of length Δ, respectively.

To compute $\alpha^l(\Delta)$ and $\alpha^u(\Delta)$, we introduce two functions $\phi^l(k)$ and $\phi^u(k)$. The former denotes the minimum number of bits constituting *any* k consecutive stream objects in a stream, and the latter denotes the corresponding maximum number of bits. In the case of an MPEG decoder, these two functions can be obtained by analyzing a number of video clips that are *representative* of the clips to be processed by the target decoder (e.g. clips having the same resolution and bitrate).

Given the functions $\phi^l(k)$ and $\phi^u(k)$, it is possible to compute the *pseudo-inverse* of these two functions, denoted by $\phi^{l^{-1}}(n)$ and $\phi^{u^{-1}}(n)$, where the argument n is the number of bits. $\phi^{l^{-1}}(n)$ and $\phi^{u^{-1}}(n)$ return the maximum and minimum number of stream objects that can be constituted by n bits respectively. Since we assume the input bit stream arrives at b at a constant rate of r bits/sec, we have:

$$\alpha^l(\Delta) = \phi^{u^{-1}}(r\Delta) \text{ and } \alpha^u(\Delta) = \phi^{l^{-1}}(r\Delta)$$

Similarly, we can characterize the variability in the number of processor cycles required to process any stream object using two functions $\gamma^l(k)$ and $\gamma^u(k)$. Both these functions take the number of stream objects k as an argument. $\gamma^l(k)$ returns the minimum number of processor cycles required to process *any* k consecutive stream objects and $\gamma^u(k)$ returns the corresponding maximum number of processor cycles.

Finally, we assume that the playout buffer B is read by the output device at a constant rate of c stream objects/sec, after a playout delay of t_{pd} seconds. Let the function $C(t)$ be the number of stream objects readout by the output device over the time interval $[0, t]$, then obviously,

$$C(t) = \begin{cases} 0 & \text{if } t \leq t_{pd} \\ c(t - t_{pd}) & \text{if } t > t_{pd} \end{cases}$$

Now, given the input bitrate r, the functions $\phi^l(k)$, $\phi^u(k)$, $\gamma^l(k)$ and $\gamma^u(k)$ characterizing the possible set of media clips to be decoded, and the function $C(t)$, we can compute the minimum processor bandwidth f_{av} to sustain the playout rate of c stream objects/sec. As mentioned in Section 2, this is equivalent to requiring that B never underflows. Let $y(t)$ denote the total number of stream objects written into B over the time interval $[0, t]$. Then the playout buffer underflow constraint is equivalent to requiring that $y(t) \geq C(t)$ for all $t \geq 0$.

Let the processor bandwidth allocated by the scheduler to the media-processing application be f Hz (cycles/sec). This results in the media stream receiving a *service* of $\beta(\Delta)$, which like $\alpha^l(\Delta)$, represents the minimum number of stream objects that are guaranteed to be processed (if available in the buffer b) within any time interval of length Δ. It can be shown that $y(t) \geq (\alpha^l \otimes \beta)(t), \forall t \geq 0$, where $\otimes$ is the *min-plus convolution*[3] operator (see [1, 5] for details). Hence, for the constraint $y(t) \geq C(t), \forall t \geq 0$ to hold, it is sufficient that the following inequality holds:

$$(\alpha^l \otimes \beta)(t) \geq C(t),\ \forall t \geq 0 \tag{1}$$

It is known from the duality between the min-plus convolution and deconvolution operators, that for any three functions f, g and h, $h \geq f \oslash g$ if and only if $g \otimes h \geq f$ (see [1] and the references therein). By applying this result to inequality (1) we obtain:

$$\beta(t) \geq (C \oslash \alpha^l)(t),\ \forall t \geq 0 \tag{2}$$

Note that $\beta(t)$ in inequality (2) is defined in terms of the number of stream objects that need to be processed within any time interval of length t. To obtain the equivalent service in terms of processor cycles, we can use the function $\gamma^u(k)$ defined above. The minimum service that needs to be guaranteed by the processor to ensure that the playout buffer never underflows is given by:

$$\gamma^u(\beta(t)) = \gamma^u((C \oslash \alpha^l)(t)) = \gamma^u(C(t) \oslash \phi^{u^{-1}}(rt)) \tag{3}$$

processor cycles for all $t \geq 0$. Hence, the minimum processor bandwidth that the scheduler needs to allocate to the media-processing task, to sustain its playout rate is given by:

$$f_{av} = \min\{f \mid ft \geq \gamma^u(\beta)(t),\ \forall t \geq 0\} \tag{4}$$

In other words, if the scheduler allocates this bandwidth then it can be guaranteed that the playout buffer B will never underflow, provided the output device starts consuming stream objects after a delay of t_{pd} time units. From Eqs. (3) and (4), it can be shown that as the playout delay t_{pd} is increased, f_{av} decreases till a certain value, after which it stabilizes. This corresponds to the "average" rate at which the stream needs to be processed to sustain the playout rate. Eqs. (3) and (4) can therefore be used to quantify the tradeoff between the playout delay (or amount of buffering) and the required processor bandwidth.

4. DYNAMIC BUFFERING

In this section we use the model proposed above to develop the schedulability test outlined in Section 2. Recall from our example in Section 2 that such a schedulability test amounts to computing feasible values of the parameters t_{fill}, t_{drain} and f_{lo}.

We first formulate the two constraints outlined in Section 2, i.e. (i) the playout buffer associated with the media-processing task should not underflow, and (ii) that the periodic task should meet its deadline. Recall that our scheduling strategy involves a cyclic repetition of two stages:

Stage 1 Once the periodic task is triggered (say at time t'), the processor bandwidth allocated to the media-processing task is increased to f_{hi} to fill up the playout buffer B. For simplicity, we assume that f_{hi} is equal to f_{max}

[3]For two functions f and g, $(f \otimes g)(t) = \inf_{0 \leq s \leq t}\{f(t-s) + g(s)\}$. Similarly, the min-plus deconvolution operator $\oslash$ is defined as $(f \oslash g)(t) = \sup_{s \geq 0}\{f(t+s) - g(s)\}$.

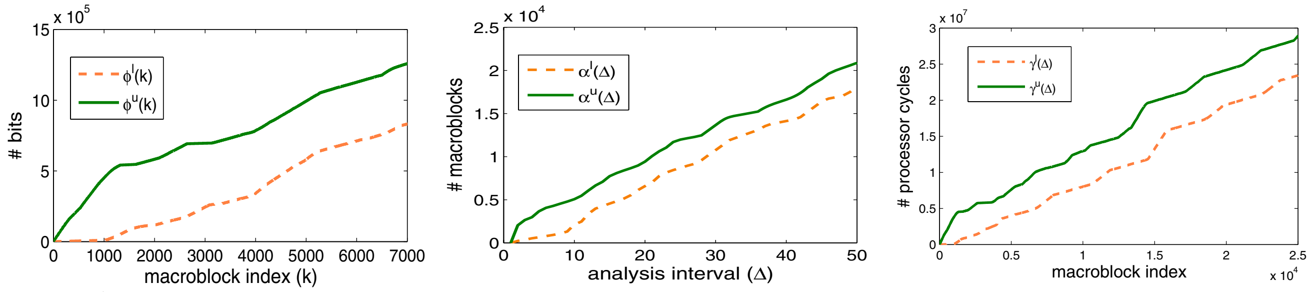

Figure 3: The functions ϕ, α and γ characterizing a high bitrate, high resolution class of video clips.

(which is the available bandwidth for running user applications). During this stage, the periodic task is suspended (if $f_{hi} < f_{max}$ then it runs with a processor bandwidth of $f_{max} - f_{hi}$). This stage lasts during the time interval $[t', t' + t_{fill})$.

Stage 2 Now the media-processing task receives a bandwidth of f_{lo} and the remaining bandwidth of $f_{max} - f_{lo}$ is allocated to the periodic task. This stage lasts during interval $[t' + t_{fill}, t' + t_{fill} + t_{drain})$.

To satisfy the buffer underflow constraint, the fill level of B should be greater than or equal to zero at time $t' + t_{fill} + t_{drain}$. In order to mathematically formulate this constraint, let β_{hi} be the *service* provided to the media-processing task when it is allocated a processor bandwidth of f_{hi}. Clearly, $\beta_{hi}(\Delta) = \gamma^{u^{-1}}(f_{hi} \cdot \Delta)$, where $\gamma^{u^{-1}}$ is the pseudo-inverse of γ^u; it takes as an argument a certain number of processor cycles and returns the *minimum* number of stream objects that are guaranteed to be processed using these cycles. Similarly, let β_{lo} denote the service corresponding to the processor bandwidth f_{lo}.

The change in the fill-level of B over the time interval $[t', t' + t_{fill} + t_{drain})$ can now be lower bounded by (i.e. the fill-level of B cannot *decrease* by more than this amount):

$$(\alpha^l \otimes \beta_{hi})(t_{fill}) - c \cdot t_{fill} \; + \; (\alpha^l \otimes \beta_{lo})(t_{drain}) - c \cdot t_{drain} \quad (5)$$

With a slight abuse of notation, let us refer to the above expression Eq. (5). The first term of this sum captures the change in fill-level over the interval $[t', t' + t_{fill})$ and the second term corresponds to the interval $[t' + t_{fill}, t' + t_{fill} + t_{drain})$. Clearly, if Eq. (5) is greater than or equal to zero, then the playout buffer underflow constraint is satisfied.

Next, we formulate the second constraint, i.e. the periodic task should meet its deadline. Let the execution requirement, period and deadline of this task be e, p and d, with $p \geq d$. Clearly, this task is schedulable if:

$$t_{fill} + t_{drain} \leq p \quad (6)$$

and

$$(f_{max} - f_{lo}) \cdot t_{drain} \geq e \quad (7)$$

Hence, our task set is schedulable if there exists t_{fill}, t_{drain} and f_{lo} for which Eq. (5) ≥ 0 and Eqs. (6) and (7) are satisfied. Unfortunately, this system of equations cannot be solved to obtain a closed-form solution to t_{fill}, t_{drain} and f_{lo}. Hence, we compute Eq. (5) for all possible values of t_{fill}, t_{drain} and β_{lo} and then identify the combinations of $(t_{fill}, t_{drain}, \beta_{lo})$ for which Eqs. (6) and (7) are satisfied. Such combinations then constitute schedulable solutions.

5. EXPERIMENTAL EVALUATION

As mentioned in Section 1, our scheme requires a one-time simulation of the applications that are to be scheduled on the mobile device. Unlike desktops, personal mobile devices typically run a small set of predefined applications. We require each of them to be simulated in isolation to determine their execution requirements. For applications that do not involve the processing of any continuous media stream (e.g. the Datebook), the parameters to be determined are their execution time, period and deadline. For media-processing applications, the parameters involved are the functions ϕ and γ, the input bitrate r and the consumption rate c by the playout device. We first describe our simulation setup that is used to obtain ϕ and γ for a high-resolution, high-bitrate class of video streams. These functions are then used to implement our schedulability analysis for a decoder application running on a mobile platform along with a periodic task.

5.1 Simulation Setup

We modelled our processor using the *sim-profile* configuration of the SimpleScalar instruction set simulator. Our media-processing task was an MPEG-2 decoder, whose source code was annotated with *start* and *stop* counters to record the number of processor cycles consumed by each stream object. To characterize the execution requirement of the decoder, we used a set of video clips having an average bitrate of 6000 kbps and a resolution of 704 × 480 pixels. The display rate of these clips was 30 fps. Figure 3 shows the three functions ϕ, α and γ for this class of video clips. Recall from Section 3.1 that ϕ characterizes the variability in the number of bits constituting each macroblock in the compressed video stream, α characterizes the variability in the arrival pattern of the video stream at the buffer b and γ captures the variability in the execution requirement of each macroblock. Clearly, such a characterization is more expressive than traditional best/worst bounds which are overly optimistic/pessimistic.

5.2 Schedulability Analysis

Recall the example described in Section 2. To estimate the processor bandwidth f_{av} occupied by the MPEG decoder for different values of playback delay t_{pd}, we use Eq. (4) from Section 3.1. Figure 4 shows how f_{av} decreases with increasing t_{pd}, starting with $t_{pd} = 10$ ms. With no other tasks running on processor, t_{pd} is typically chosen to be a relatively small value. However, Figure 4 shows the potential for decreasing the allocated bandwidth to the decoder by increasing the fill-level of the playout buffer B.

Let us now consider a setup where the MPEG-2 decoder concurrently runs with a periodic task on a 510 MHz processor (bandwidth available for user applications). The periodic task is characterized by a period of 500 ms, which is also equal to its deadline and has an execution requirement of 100×10^6 cycles. Hence, $f_{hi} = 510$ MHz. We would like

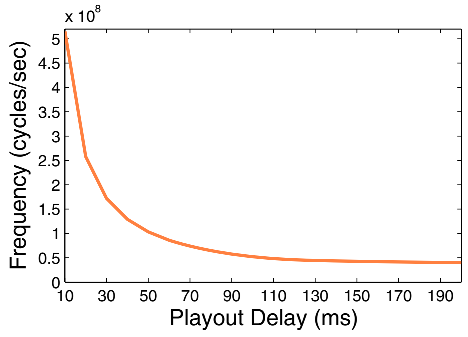

Figure 4: Buffering time versus workload.

to estimate if this system is schedulable, i.e. whether there exists feasible t_{fill}, t_{drain} and f_{lo}.

Figure 5 shows the value of Eq. (5) for this setup for different values of t_{fill} and t_{drain} with f_{lo} set to zero. Note that the vertical axis (z-axis) of this plot corresponds to the number of (excess) decompressed macroblocks in the playout buffer after one cycle of $t_{fill} + t_{drain}$. The region of this plot that is above the plane $z = 0$ corresponds to values of t_{fill} and t_{drain} for which the playout buffer does not underflow. This region is labelled as the *feasible region*. The region below $z = 0$ corresponds to values of t_{fill} and t_{drain} for which the playout buffer underflows (i.e. its fill level decreases after the $t_{fill} + t_{drain}$ cycle). This region is labelled as the *infeasible region*. Clearly, we are interested in the feasible region. However, this region (or a subset of it) also has to satisfy the constraints given by Eqns. (6) and (7) for the periodic task to be schedulable. The subset of the feasible region that satisfies these two equations is labelled as the *schedulable region*.

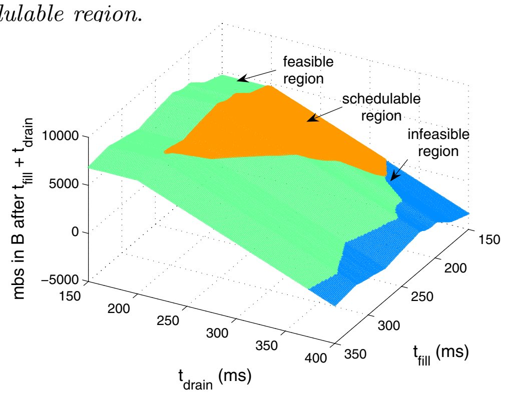

Figure 5: Schedulability region of a schedulable system.

Finally, Figure 6 shows the schedulability regions for three different values of f_{lo}. The lowermost surface corresponds to $f_{lo} = 0$ MHz, the middle surface corresponds to $f_{lo} = 50$ MHz and the topmost corresponds to $f_{lo} = 100$ MHz. All other parameters, such as f_{hi} and those describing the periodic task remain the same as before. Note from Figure 6 that the system is schedulable for all these three values of f_{lo}. However, the schedulable values of t_{fill} and t_{drain} change with different values of f_{lo}. It may be noted that different values of t_{fill} and t_{drain} are also associated with different scheduling overheads and buffer requirements. Our model offers the possibility of quickly visualizing the design space for selecting the appropriate scheduler parameters.

6. CONCLUDING REMARKS

Our proposed technique can be exploited to enhance the schedulability of media-processing applications when concurrently run with other applications that can be modelled as periodic tasks. The underlying idea was to accurately model the variability in the processing requirements of media-processing applications and appropriately use buffering to periodically free up a portion of the processor's bandwidth to support other tasks. Our results can be useful for designing and tuning application-specific schedulers for personal mobile devices which run a restricted set of applications.

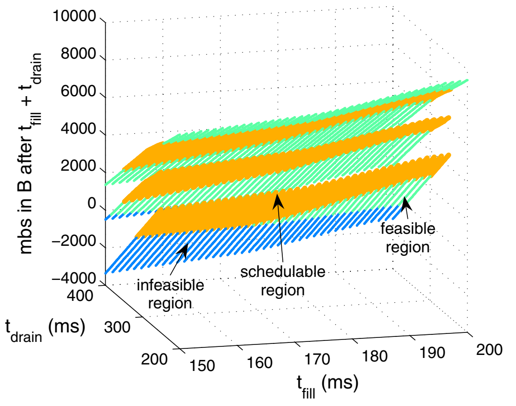

Figure 6: Schedulability regions for different f_{low}.

Acknowledgements: We thank Wei Tsang Ooi for the numerous helpful discussions that led to this paper.

7. REFERENCES

[1] J.-Y. Le Boudec. Some properties of variable length packet shapers. *IEEE/ACM Transactions on Networking*, 10(3):329–337, 2002.

[2] L. Cai and Y.-H. Lu. Energy management using buffer memory for streaming data. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(2):141–152, 2005.

[3] C.-F. Chiasserini and R. R. Rao. Improving battery performance by using traffic shaping techniques. *IEEE Journal on Selected Areas in Communications*, 19(7):1385–1394, 2001.

[4] K. J. Duda and D. R. Cheriton. Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler. In *ACM Symposium on Operating System Principles (SOSP)*, 1999.

[5] A. Elwalid and D. Mitra. Traffic shaping at a network node: Theory, optimum design, admission control. In *INFOCOM*, 1997.

[6] L. Georgiadis, R. Guérin, V. G. J. Peris, and K. N. Sivarajan. Efficient network QoS provisioning based on per node traffic shaping. *IEEE/ACM Transactions on Networking*, 4(4):482–501, 1996.

[7] P. Goyal, X. Guo, and H. M. Vin. A hierarchical CPU scheduler for multimedia operating systems. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 1996.

[8] S. Heithecker and R. Ernst. Traffic shaping for an FPGA based SDRAM controller with complex QoS requirements. In *DAC*, 2005.

[9] J. Hu and Y.-H. Lu. Buffer management for power reduction using hybrid control. In *IEEE Conference on Decision and Control and the European Control Conference*, 2005.

[10] S. Manolache, P. Eles, and Z. Peng. Buffer space optimisation with communication synthesis and traffic shaping for NoCs. In *DATE*, 2006.

[11] C. Poellabauer and K. Schwan. Energy-aware traffic shaping for wireless real-time applications. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2004.

[12] M.J. Rutten, J.T.J. van Eijndhoven, and E.-J.D. Pol. Design of multi-tasking coprocessor control for eclipse. In *CODES*, 2002.

[13] G. Varatkar and R. Marculescu. On-chip traffic modeling and synthesis for MPEG-2 video applications. *IEEE Transactions on VLSI*, 12(1):108–119, January 2004.

[14] E. Wandeler, A. Maxiaguine, and L. Thiele. Performance analysis of greedy shapers in real-time systems. In *DATE*, 2005.