Approximate VCCs: A New Characterization of Multimedia Workloads for System-level MpSoC Design

Yanhong Liu Samarjit Chakraborty Wei Tsang Ooi
Department of Computer Science
National University of Singapore
{liuyanho,samarjit,ooiwt}@comp.nus.edu.sg

ABSTRACT

System-level design methods specifically targeted towards multimedia applications have recently received a lot of attention. Multimedia workloads are known to have a high degree of variability. Therefore, designs based on a worst-case analysis of such workloads tend of be overly pessimistic. We address this issue by introducing a new concept called *approximate variability characterization curves* (or Approximate VCCs), to characterize the "average-case" behavior of multimedia workloads in a parameterized fashion. Since most multimedia applications only have soft real-time constraints, it is often possible to tolerate a small amount of performance degradation. By allowing such small degradations in the performance, large amounts of resource savings are possible. The concept of Approximate VCCs that we present in this paper allows a designer to quantitatively account for the performance degradation and the associated resource savings. We illustrate this using two typical system design cases.

Categories and Subject Descriptors

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

General Terms

Performance, Design

Keywords

Multimedia, Workload, System-level design

1. INTRODUCTION

Today multimedia applications run on a wide range of consumer electronic devices, ranging from set-top boxes to PDAs and mobile phones. Due to factors like flexibility, low design costs and high time-to-market pressures, many of these devices are now designed using configurable multiprocessor System-on-Chip (MpSoC) platforms. Examples of such platforms are the Eclipse architecture template and the Viper SoC architecture from Philips that target advanced set-top boxes and DTVs, OMAP from Texas Instruments and PrimeXsys from ARM. To configure a platform architecture for a specific multimedia application, a common practice is to use a set of representative audio/video clips that would be processed by the application. The workload generated by such a representative set is then used to determine parameters such as on-chip buffer sizes, clock speeds of the different processors, bus widths and cache configurations.

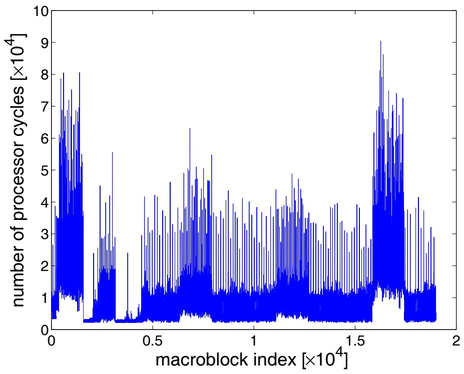

Figure 1: Processor cycle requirements of a sequence of macroblocks for an MPEG-2 decoder application.

Multimedia workloads are known to exhibit a high variation in their resource demands. For example, the ratio of the worst-case and the average load on a processor running a multimedia task can easily be as high as a factor of 10 [12]. On the other hand, multimedia applications typically have soft real-time constraints. This allows certain tasks to miss their deadlines or a few data items to be occasionally dropped from a buffer, without significantly deteriorating the output quality. A consequence of the above two characteristics is that a worst-case analysis of multimedia workloads often lead to overly pessimistic results. At the same time, a straightforward average-case analysis does not suffice because of the high variability in the workload. Hence, appropriately characterizing multimedia workloads for system-level design is a tricky problem. Figure 1 shows the processor cycle requirements of a sequence of macroblocks for an MPEG-2 decoder application. The large variation in the processor cycle requirements for the different macroblocks is clearly noticeable.

To address the above problem, in this paper we propose a new characterization of multimedia workloads, that can be used to characterize the "average-case" behavior of a workload in a parameterized fashion. Towards this, we take into account the *frequency* with which the worst-case occurs and

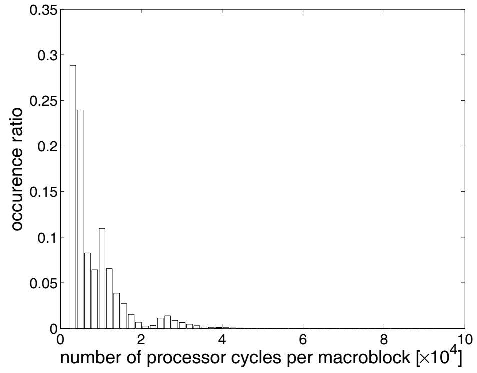

Figure 2: Histogram of the processor cycle demand per macroblock for an MPEG-2 video. The minimum and the maximum cycle demands are 2218 **and** 92247 **respectively.**

discard worst-case scenarios that do not occur frequently enough. Therefore, what we refer to as "average-case" (for the sake of simplicity), actually denotes the worst-case that occurs often enough. We quantify "often enough" using a parameter that is specified by the designer. By ignoring worst-case scenarios that do not occur very frequently, significant amounts of resource savings are usually possible, with negligible loss in the output audio/video quality. Our proposed characterization can also be used to quickly identify the tradeoffs between the output quality and the potential resource savings. Using purely simulation-oriented techniques to determine such tradeoffs is not only expensive in terms of the simulation time involved, but is often also impractical.

The concept of VCCs was proposed in [9, 10] to characterize different kinds variabilities occurring in multimedia workloads. These include (i) data-dependent variability in the execution time of multimedia processing tasks, (ii) variability in the amount of input and output data produced and consumed by a task (for example, the variable length decoding task in MPEG decoders consume a variable number of bits for each processed macroblock), and (iii) burstiness in on-chip traffic arising out of multimedia processing on multiprocessor architectures [13]. VCCs characterize *best-* and *worst-case* scenarios without considering the frequency with which such scenarios occur. Simulating the execution of an MPEG-2 decoder with a randomly chosen video clip shows that the worst-case processor cycle demand to decode a macroblock occurs in about 0.02% of the total number of macroblocks processed. For the execution trace in Figure 1, the histogram of the processor cycle demand per macroblock is shown in Figure 2. From this figure, it may be noted that the cycle demands of about 90% of all the macroblocks are less than half of the maximum/worst-case cycle demand of a macroblock. VCCs, as proposed in [10], would record this maximum value without taking into account the frequency of its occurrence.

An approximate VCC or ε-VCC ignores at most ε percentage of the data from the right-hand side of the histogram in Figure 2. The remaining data is then used to compute the worst-case scenario. As a result, all worst-case scenarios, whose cumulative frequency of occurrence is less than ε percent are ignored. Given a trace such as the one shown in Figure 1, we show how to bound the error corresponding to different values of ε, for typical system-level design problems. An example of this is to bound the maximum number of data items that may be dropped from a buffer, when the buffer sizing is done based on ε-VCCs. It may be noted that when buffer sizing is based on VCCs, it can be guaranteed that no data items will be dropped [10], albeit at the cost of much larger buffer sizes compared to when ε-VCCs are used. Since worst-case scenarios occur very infrequently (as discussed above), *significant savings* are achieved by using ε-VCCs, at the cost of *negligible loss* in output audio/video quality.

Related work: The concept of VCCs has its foundations in the theory of *network calculus* [4, 7]. Whereas the originally proposed network calculus may be seen as a deterministic queuing theory for analyzing communication networks, recently a number of extensions to this theory have been developed [3, 6]. These extensions are concerned with providing statistical service guarantees rather than deterministic guarantees, which often lead to resource over-provisioning. Along similar lines, Ayyorgun and Cruz [1, 2] have recently proposed a service model which allows a certain portion of network packets to be dropped based on a loss parameter. In contrast to the work presented in this paper, they, however, concentrate on a multiplexing problem and study the necessary capacity of a multiplexer to provide deterministic service guarantees to each flow passing through it. As we already mentioned, all the above efforts focus only on the domain of communication networks, and the results obtained can not be applied to our problem setup (multimedia processing on MpSoC platforms) in any straightforward manner.

Within the embedded systems domain, the concept of *Stochastic Automatic Networks* (SANs) [11] has been proposed for average-case performance analysis of platform architectures. Whereas this is an automata-theoretic formalism, the workload characterization that we present here is purely "functional", where the "state" of the system is not modeled. The focus is primarily on modeling the variability in the arrival process and the execution demand of multimedia streams, rather than the *state* of the system processing these streams. We believe that there is a potential for integrating our work with the SAN formalism.

Organization of the paper: Section 2 describes our model of MpSoC platforms and the concept of VCCs. This is followed by our definition of ε-VCCs. In Section 3 we present an analytical method for bounding the error incurred while designing a system based on ε-VCCs. Experimental results which validate our method are presented in Section 4.

2. MULTIMEDIA WORKLOAD CHARACTERIZATION

2.1 MpSoC Platform Model

We consider the following system-level view of multimedia stream processing on an MpSoC platform. The platform architecture consists of multiple processing elements (PEs) onto which different parts of an application are mapped. An input multimedia stream enters a PE, gets processed by the task(s) implemented on this PE, and the processed stream enters another PE for further processing. At the input of each PE is a buffer, which is a FIFO channel of fixed capacity, and is used to store the incoming stream to be processed. Finally, the fully processed stream is written into a *playout buffer* which is read by some *real-time client* (RTC)

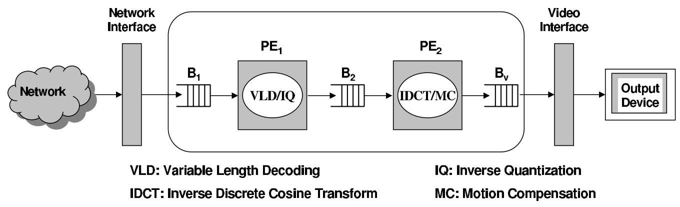

Figure 3: An MPEG-2 decoder on an MpSoC platform.

such as an audio or a video output device. For the sake of generality, we consider any multimedia stream to be made up of a sequence of *stream objects*. A stream object might be a bit belonging to a compressed bitstream representing a coded video clip, or a macroblock, or a video frame, or an audio sample—depending on where in the architecture the stream exists.

As an example, Figure 3 shows an architecture with two PEs (PE_1 and PE_2), implementing an MPEG-2 decoder application. The *variable length decoding* (VLD) and *inverse quantization* (IQ) tasks have been mapped onto PE_1, and the *inverse discrete cosine transform* (IDCT) and *motion compensation* (MC) tasks onto PE_2. A video stream, after being downloaded over a network, enters the buffer B_1, which is read by PE_1. The resulting partially decoded macroblocks are then written into B_2, which is read by PE_2. The fully decoded macroblocks are written into the playout buffer B_v, which is read by the video output device at a prespecified rate.

Typical constraints that are associated with the above setup are: none of the buffers should overflow and the playout buffer should not underflow. Configuring the above architecture, such as determining the minimum sizes of the buffers B_1, B_2 and B_v is difficult because of the complex nature of multimedia workloads. As we already mentioned in the previous section, the execution requirements of different macroblocks might be highly variable. Further, the bitstream arriving over the network might also be bursty.

2.2 Variability Characterization Curves

VCCs can be used to quantify best-case and worst-case characteristics of *sequences*. These can be sequences of consecutive stream objects belonging to a stream, or sequences of consecutive time intervals of some specified length. A VCC $\mathcal{V}$ is defined as a tuple ($\mathcal{V}^l(k)$, $\mathcal{V}^u(k)$), where k represents the length of the sequence. Let the function P be a measure of some property over a sequence. If $P(n)$ denotes the measure of this property for the first n items of the sequence, then $\mathcal{V}^l(k)$ and $\mathcal{V}^u(k)$ for all $k \geq 0$ are defined as follows.

$$\begin{aligned} \mathcal{V}^l(k) &= \inf_{i\geq 0}\{P(i+k) - P(i)\} \\ \mathcal{V}^u(k) &= \sup_{i\geq 0}\{P(i+k) - P(i)\} \end{aligned} \quad (1)$$

$\mathcal{V}^l(k)$ and $\mathcal{V}^u(k)$ therefore provide lower and upper bounds on the measure P, for *all* subsequences of length k, within a larger sequence. Let us now consider a few concrete examples of VCCs.

Workload Curve $\gamma = (\gamma^l, \gamma^u)$: The VCC γ is used to characterize the variability in the execution requirements of a sequence of stream objects to be processed by a PE. In this case, given a sequence of stream objects, $P(n)$ denotes the total number of processor cycles required to process the first n stream objects. Hence, $\gamma^l(k)$ and $\gamma^u(k)$ denote the minimum and the maximum number of processor cycles that might be required by *any* k consecutive stream objects within the given sequence.

Let $e_{\min}$ and $e_{\max}$ be the minimum and the maximum number of processor cycles required by any single stream object belonging to a sequence. In the execution trace shown in Figure 1, for any reasonably large value of k, $\gamma^l(k)$ is clearly greater than $k \times e_{\min}$. Further, the difference between them increases with increasing k. Similarly, $\gamma^u(k)$ is clearly smaller than $k \times e_{\max}$. Hence, the VCC γ is more expressive compared to simple best- or worst-case characterizations commonly used in the real-time systems domain.

It is also meaningful to construct a *pseudo-inverse* of a VCC $\mathcal{V}$, which we denote as $\mathcal{V}^{-1}$. In the case of a workload curve, $\gamma^{l^{-1}}(e) = \min_{k\geq 0}\{k \mid \gamma^l(k) \geq e\}$ and $\gamma^{u^{-1}}(e) = \max_{k\geq 0}\{k \mid \gamma^u(k) \leq e\}$. Hence, $\gamma^{l^{-1}}(e)$ denotes the maximum number of stream objects that may be processed using e processor cycles. $\gamma^{u^{-1}}(e)$ denotes the minimum number of stream objects that are guaranteed to be processed using e processor cycles.

Arrival Curve $\alpha = (\alpha^l, \alpha^u)$: This VCC is used to characterize the burstiness in the arrival pattern of stream objects. Given a trace of the arrival times of a sequence of stream objects (e.g. the partially processed macroblocks being written into the buffer B_2 in Figure 3), $\alpha^l(\Delta)$ and $\alpha^u(\Delta)$ denote the minimum and the maximum number of stream objects that arrive within *any* time interval of length Δ. For notational simplicity, henceforth we will denote the pseudo-inverse of α (i.e. α^{-1}) as ξ.

Service Curve $\beta = (\beta^l, \beta^u)$: Due to the variability in the execution requirements of stream objects, the number of stream objects that can potentially be processed within any specified time interval varies (even when the processor runs at a constant frequency). We will use $\beta^l(\Delta)$ and $\beta^u(\Delta)$ to denote the minimum and the maximum number of stream objects that can be processed (or served) by a processor within *any* time interval of length Δ. β^l and β^u may also be derived from a trace of execution requirements of stream objects (such as the one shown in Figure 1) and the clock frequency with which the processor is being run.

2.3 Approximate VCCs

The use of VCCs to analyze and tune platform architectures for multimedia processing has been illustrated in [10]. However, in the above formulation, the best- and worst-case characterization using VCCs do not take into account the frequency with which the best- or the worst-case occurs. Approximate VCCs generalize the concept of VCCs and take into account the frequency with which the best-/worst-case occurs.

Recall our definition of VCCs, as given by Eqn. (1). Now, for any given k, let a set S be defined as follows: $S = \{P(i+k)-P(i) \mid i \geq 0\}$. Instead of computing the minimum and maximum value in the multiset S, to compute ε-VCCs, we first remove certain extreme observations from S and then compute the minimum and the maximum value from the remaining elements.

Let S^l_ε denote the set resulting from removing the smallest ε percent of items from the set S. Similarly, S^u_ε denotes the set resulting from removing the largest ε percent of items from S. An ε-VCC $\mathcal{V}_\varepsilon$ can now be defined as follows: $\mathcal{V}^l_\varepsilon(k) = \inf_{i\geq 0}\{S^l_\varepsilon\}$ and $\mathcal{V}^u_\varepsilon(k) = \sup_{i\geq 0}\{S^u_\varepsilon\}$.

The above definition of ε-VCC implies that ε percent of items in S are less than $\mathcal{V}^l_\varepsilon$ and ε percent of items in S are larger than $\mathcal{V}^u_\varepsilon$. Since the set S can contain a poten-

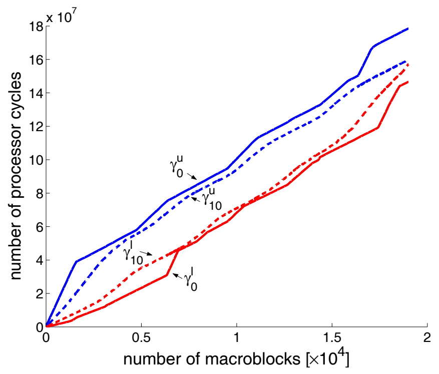

Figure 4: Approximate workload curves.

tially large number of elements, a computationally efficient algorithm is necessary to compute $\mathcal{V}_\varepsilon^l$ and $\mathcal{V}_\varepsilon^u$. We adopt a histogram-based algorithm [14] which is simple and efficient. Although the results obtained are not as accurate as percentile-based methods [5], they are sufficiently precise for the problem setups that we are interested in.

The histogram-based algorithm works as follows. Let D_{min} and D_{max} be the minimum and the maximum values of the elements in S. Suppose that the range $[D_{min}, D_{max}]$ is split into n equal-sized bins with the bin boundaries being $c_0, c_1, \cdots, c_n$. First, we construct a histogram for all the elements in S. We then compute r_i (for all $1 \leq i \leq n$), which is the ratio of the number of elements in the i-th bin $(c_{i-1}, c_i]$ to the total number of elements in S. Clearly, the sum $\sum_{j=1}^{i} r_j$ represents the fraction of items which are not larger than c_i. We then define a function F, where $F(c_i) = \sum_{j=1}^{i} r_j$ for $0 \leq i \leq n$ (note that F is defined only for these values). Finally, $\mathcal{V}_\varepsilon^l$ and $\mathcal{V}_\varepsilon^u$ are defined as follows.

$$\begin{aligned} \mathcal{V}_\varepsilon^l(k) &= \max_{0 \leq i \leq n}\{c_i \mid F(c_i) \leq \tfrac{\varepsilon}{100}\} \\ \mathcal{V}_\varepsilon^u(k) &= \min_{0 \leq i \leq n}\{c_i \mid F(c_i) \geq 1 - \tfrac{\varepsilon}{100}\} \end{aligned} \quad (2)$$

It follows from the above definition that VCCs are a special case of ε-VCCs, with ε set to zero. Figure 4 shows an approximate workload curve (for the VLD/IQ task in Figure 3) with $\varepsilon = 10$. The same figure also shows the corresponding workload curve (i.e. the case where ε is set to 0). It can clearly be seen that the approximate workload curves represent more conservative bounds on the execution requirements of sequences of stream objects, compared to the lower and upper bounds obtained from the (exact) workload curves.

3. ERROR ANALYSIS

In a typical system design process, a designer would analyze a set of representative audio/video clips to obtain different ε-VCCs. These ε-VCCs would represent the workload that the system will be required to support. In the context of platform-based design, these ε-VCCs would determine different platform configuration parameters such as sizes of on-chip buffers, bus widths and clock frequencies of the different on-chip processors. Since ε-VCCs represent more conservative bounds and ignore infrequent best- and worst-cases, the resulting systems can also be more conservatively designed (and hence would be less expensive), albeit at the cost of small errors. For example, the minimum on-chip buffer sizes determined using ε-VCCs would be smaller compared to those determined using VCCs. The difference in size would depend on the value of ε chosen. However, the savings would come at the cost of occasionally some stream objects being dropped from the buffer. In this section we present an analytical method that can be used to bound the error incurred for any ε. We present this method in the context of two system design problems: optimal on-chip buffer sizing and processor frequency selection.

3.1 On-Chip Buffer Sizing

Consider a PE (such as PE_2 in Figure 3) processing a stream whose arrival process is bounded by the arrival curve α. Let β be the service curve offered by the PE. It can then be shown that the minimum size of the buffer (or the maximum backlog) at the input of this PE (i.e. B_2 in this case) is equal to $\sup_{\Delta \geq 0}\{\alpha^u(\Delta) - \beta^l(\Delta)\}$. Let us denote this as $b_{\max}$ (the maximum backlog).

To see how β^l is obtained, let us assume that the PE runs at a clock frequency of f clock cycles/second. Given a trace of processor cycle requirements per stream object (such as the one shown in Figure 1) it is possible to compute the workload curve γ^u. It is then easy to see that $\gamma^{u^{-1}}(f \cdot \Delta)$ is the minimum number of stream objects that are guaranteed to be processed within any time interval of length Δ. Hence, we set $\beta^l(\Delta)$ to be equal to $\gamma^{u^{-1}}(f \cdot \Delta)$.

For the buffer sizing to be done using ε-VCCs, we proceed as follows. Instead of using the arrival curve α directly, we use its pseudo-inverse ξ. From a representative trace of arrival times of a sequence of stream objects, we compute $\xi_\varepsilon^l(k)$. From the trace of execution time requirements of the stream objects we compute β^l, as described above. The estimated maximum backlog is then given by:

$$b_\varepsilon = \sup_{k \geq 0}\{k - \beta^l(\xi_\varepsilon^l(k))\}$$

It may be shown that $\sup_{k \geq 0}\{k - \beta^l(\xi_\varepsilon^l(k))\}$ is equal to $\sup_{\Delta \geq 0}\{\alpha_{\varepsilon'}^u(\Delta) - \beta^l(\Delta)\}$ (which is similar in form to the computation of $b_{\max}$ shown above). Here, $\alpha_{\varepsilon'}^u(\Delta)$ is obtained by inverting $\xi_\varepsilon^l(k)$. It may be noted that by inverting $\xi_\varepsilon^l(k)$ we obtain an approximate arrival curve whose *approximation ratio* ε' is different from the approximation ratio ε of $\xi_\varepsilon^l(k)$.

Clearly, if the buffer size is set to b_ε then stream objects might occasionally be dropped. Given a trace of arrival times of stream objects at the buffer, we can bound the maximum number of stream objects that might be dropped. We assume that β^l, which was obtained from a set of representative multimedia streams, also holds for this trace (i.e. $\beta^l(\Delta)$ is the minimum number of stream objects that are guaranteed to be processed within any time interval of length Δ, for this stream as well).

Let $T(i)$ denote the arrival time of the i-th stream object at the buffer. Let $\xi(i,k) = T(i) - T(i-k)$ denote the length of the time interval during which the previous k consecutive stream objects adjacent to the i-th stream object arrive ($0 \leq k \leq i$). Then $\beta^l(\xi(i,k))$ represents the minimum number of stream objects that the PE can process during this time interval.

THEOREM 1. *The maximum backlog when the i-th stream object arrives at the buffer is equal to*

$$\sup_{0 \leq k \leq i}\{k - \beta^l(\xi(i,k))\}$$

Hence, the i-th stream object might be dropped if

$$\sup_{0 \leq k \leq i}\{k - \beta^l(\xi(i,k))\} > b_\varepsilon$$

In the above inequality, the value of $\beta^l(\xi(i,k))$ is estimated to be $\gamma^{u^{-1}}(f \cdot \xi(i,k))$. This assumes that the $\beta^l(\xi(i,k))$ consecutive stream objects processed within the time interval of length $\xi(i,k)$ require the maximum possible number of processor cycles. If we instead use the approximate upper workload curve γ_ε^u, then the above inequality may be reformulated as:

$$\sup_{0 \le k \le i} \{k - \beta_{\varepsilon'}^l(\xi(i,k))\} > b_\varepsilon$$

However, unlike the previous case, in this case we can not provide deterministic guarantees on the maximum number of dropped stream objects.

3.2 Processor Frequency Selection

Let us consider the PE PE_2 in Figure 3. The stream objects processed by it are written out into the playout buffer B_v. This buffer is read by the real-time video output device at a prespecified rate. One of the design constraints while configuring this platform architecture is to ensure that B_v never underflows. Clearly, the clock frequency of PE_2 should at least be equal to sustain the rate at which stream objects are being consumed by the output device. However, because of the variability in the execution time requirements of stream objects, computing this minimum clock frequency is not trivial. The problem becomes more complicated because of the buffering at the playout buffer. The problem of computing this frequency has become especially interesting with the advent of processor soft cores, which allow a high degree of customization. This problem was addressed in [9] using VCCs as a means of workload characterization.

Clearly, using ε-VCCs, the computed frequency will be substantially lower compared to that obtained using VCCs. For the sake of simplicity, here we have only considered the problem of computing the minimum constant frequency at which the PE needs to be run. However, the method presented in [9] can be used in the case of frequency-scalable processors as well (to compute the different frequency levels and the frequency range that the PE should support). Due to space constraints, we do not present any further details here. Our experiments results in Section 4.2 show how the minimum frequency changes with different values of ε and the bounds on the error incurred.

4. EMPIRICAL VALIDATION

To validate our scheme for workload characterization, we experimented with the setup shown in Figure 3. For our experiments, the granularity of a stream was chosen to be a macroblock.

We experimented with multiple representative video clips chosen from a set of clips, all of which have the same long-term playback rate, i.e. the same number of macroblocks are consumed per second by the video output device. For each video clip, we first used the SimpleScalar instruction set simulator to obtain traces of execution times for the VLD/IQ and IDCT/MC tasks of the MPEG-2 decoder application. We then simulated the platform architecture shown in Figure 3 using a transaction-level model of the architecture written in SystemC. Traces containing the arrival times of the macroblocks at each on-chip buffer and the buffer backlogs were obtained. The VCCs and the ε-VCCs were measured from the collected execution traces. In the following, we assume that the traces of execution times have already been obtained.

4.1 Buffer Sizing

The results reported below only concern the buffer at the input of PE_2 (i.e. B_2). Both PE_1 and PE_2 were configured to run with their long-term average frequencies. These frequencies were computed by taking into account the long-term playback rate of the output device and the average cycle demands per macroblock for the tasks implemented on them. The system was initially simulated for all the (representative) video clips, from which we obtained the approximate lower pseudo-inverse curve $\xi_\varepsilon^l(k)$ corresponding to the arrival process of stream objects at the buffer B_2. From the simulation results we also obtained the approximate upper workload curve $\gamma_\varepsilon^u(k)$ for PE_2. We then computed the buffer size b_ε. As shown in Figure 5, the computed buffer size decreases as ε is increased from 0 to 20. We observed more than 20% reduction in the buffer size when ε was set to be 5.

For each video clip, we analytically estimated the upper bound on the percentage of dropped macroblocks when the size of B_2 was set to b_ε. At the same time, we simulated the execution of this clip with the size of B_2 set be b_ε. The simulation results showed that our analytical method gives an upper bound on the percentage of dropped macroblocks for any of the clips used. Figure 6 shows the analytical bounds and simulation results for a representative video clip. We can observe that the drop ratio is upper bounded at about 5% with ε equal to 5. However, there is more than 20% reduction in the buffer size compared to when ε is equal to 0. As shown in Figure 5, we also measured the *Peak Signal-to-Noise Ratio* (PSNR) for this video clip corresponding to each buffer size. PSNR is commonly used to measure the quality of a reconstructed frame with macroblock loss, compared to the decoded frame without any loss. We defined the PSNR of a video clip as the average value of PSNRs over all those frames which suffered loss of macroblocks. Although we applied only a simple error concealment mechanism (a dropped macroblock just takes the value of the corresponding macroblock from the previous frame), Figure 5 shows that at $\varepsilon = 5$, the PSNR remains at 39.2 dB. PSNR values above 38 dB are generally accepted as good video quality [8].

4.2 Frequency Selection

We will use PE_2 to illustrate how the processor's clock frequency may be lowered if ε-VCCs are used. Based on the approximate upper workload curve γ_ε^u on PE_2 and the long-term playback rate, we computed the clock frequency f_ε for PE_2. As shown in Figure 7, considerable reduction in the frequency values were achieved when the approximate curves were used. For example, there was nearly a 20% reduction in the frequency when ε was set to be 60.

PE_1 was configured to its long-term average frequency. An initial simulation of the system was conducted for all the representative video clips, after which we had the necessary traces for the error analysis. For each video stream, we computed an upper bound on the percentage of macroblocks that can potentially miss their deadlines when PE_2 is run at different clock frequencies. When compared with simulation results, it may be seen that our analytical method gives an upper bound on the percentage of macroblocks that missed their deadlines. Table 1 shows the analytical bounds and the results obtained using simulation for a representative video clip with two different playback delay settings t_d. It may

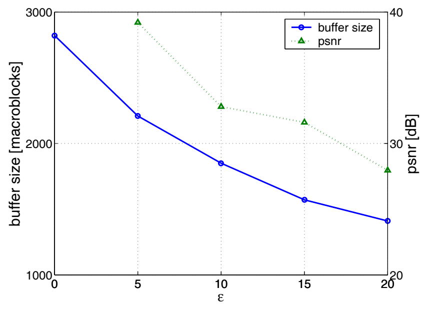

Figure 5: Computed buffer sizes for different values of ε.

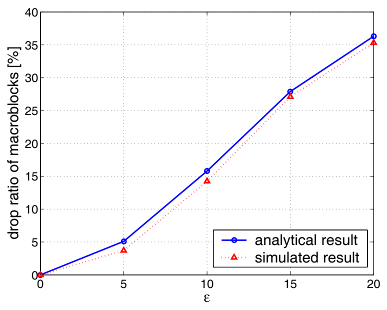

Figure 6: Percentage of macroblocks dropped from B_2 for different values of ε.

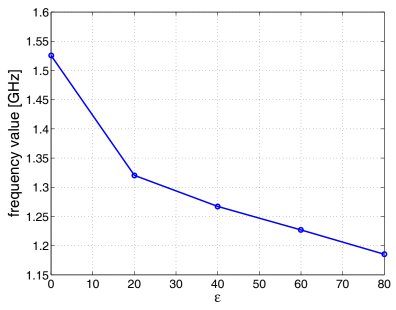

Figure 7: Frequency values of PE_2 for different values of ε.

ε	% of macroblocks missing deadlines			
	$t_d = 0.28s$		$t_d = 0.30s$	
	analysis	simulation	analysis	simulation
0	3.84	3.80	0.00	0.00
20	9.73	9.63	0.00	0.00
40	16.7	16.5	0.00	0.00
60	44.0	43.7	0.00	0.00
80	97.0	97.0	80.4	69.5

Table 1: Analytical bounds and simulation results on the percentage of macroblocks that miss their deadlines, for different values of ε.

be noted that when the delay was set to $0.30s$, none of the macroblocks missed their deadlines, even with ε set to 60, while the required frequency was reduced by nearly 20%.

5. CONCLUDING REMARKS

In this paper we proposed a parameterized scheme for characterizing multimedia workloads, based on the novel concept of *approximate variability characterization curves* or ε-VCCs. Since most multimedia applications only require soft real-time guarantees, we demonstrated that by using ε-VCCs to design and configure platform architectures, significant resource savings may be achieved with only a negligible loss in output quality.

To bound the error incurred by using ε-VCCs, we required a trace of the arrival times of stream objects to compute the maximum number of stream objects that might get dropped from a buffer. A more elegant scheme would be to provide guarantees for a *class* of arrival patterns (such as that captured by an arrival curve). Note that our scheme can provide guarantees for a class of resource requirements (i.e. those that are captured by a workload curve).

Acknowledgements: The work reported here has been partially funded by the NUS URC grant R-252-000-190-112, through the project "ASTRA: System-Level Design and Analysis of Architectures for Streaming Applications".

6. REFERENCES

[1] S. Ayyorgun and R. L. Cruz. A composable service model with loss and a scheduling algorithm. In *INFOCOM*, Hong Kong, China, March 2004.

[2] S. Ayyorgun and R. L. Cruz. A service-curve model with loss and a multiplexing problem. In *ICDCS*, Tokyo, Japan, March 2004.

[3] R. Boorstyn, A. Burchard, J. Leibeherr, and C. Oottamakorn. Statistical service assurances for traffic scheduling algorithms. *IEEE Journal on Selected Areas in Communications*, 18(13):2651–2664, 2000.

[4] J.-Y. Le Boudec and P. Thiran. *Network Calculus - A Theory of Deterministic Queuing Systems for the Internet*. LNCS 2050, 2001.

[5] W. Chase and F. Bown. *General Statistics*. John Wiley & Sons, 1997.

[6] F. Ciucu, A. Burchard, and J. Liebeherr. A network service curve approach for the stochastic analysis of networks. In *ACM Sigmetrics*, 2005.

[7] R. Cruz. A calculus for network delay, Parts 1 & 2. *IEEE Transactions on Information Theory*, 37(1), 1991.

[8] C. A. Gonzales, H. Yeo, and C. J. Kuo. Requirements for motion-estimation search range in MPEG-2 coded video. *IBM Journal of Research and Development*, 43(4), 1999.

[9] Y. Liu, A. Maxiaguine, S. Chakraborty, and W. T. Ooi. Processor frequency selection for SoC platforms for multimedia applications. In *RTSS*, Lisbon, Portugal, December 2004.

[10] A. Maxiaguine, Y. Zhu, S. Chakraborty, and W.-F. Wong. Tuning SoC platforms for multimedia processing: Identifying limits and tradeoffs. In *CODES+ISSS*, Stockholm, Sweden, September 2004.

[11] A. Nandi and R. Marculescu. System-level power/performance analysis for embedded systems design. In *DAC*, Las Vegas, Nevada, USA, June 2001.

[12] M.J. Rutten, J.T.J. van Eijndhoven, E.G.T. Jaspers, P. van der Wolf, O.P. Gangwal, and A. Timmer. A heterogeneous multiprocessor architecture for flexible media processing. *IEEE Design & Test of Computers*, 19(4):39–50, July-August 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] W. Yuan and K. Nahrstedt. Energy-efficient soft real-time CPU scheduling for mobile multimedia systems. In *SOSP*, NY, USA, October 2003.