

Generalized Rate Analysis for Media-Processing Platforms

Yanhong Liu¹ Samarjit Chakraborty¹ Radu Marculescu²

¹Department of Computer Science, National University of Singapore

²Department of Electrical & Computer Engineering, Carnegie Mellon University

E-mail: {liuyanho, samarjit}@comp.nus.edu.sg, radum@ece.cmu.edu

Abstract

In this paper we address the “rate analysis” problem for media-processing platforms consisting of multiple processor cores connected in a pipelined fashion. More precisely, we aim at determining tight bounds on the rates at which multimedia streams can be fed into such architectures. These bounds depend on architectural constraints (e.g. the available on-chip memory, bus arbitration policies, etc.), as well as the application characteristics (e.g. application partitioning and mapping, workload rates generated by different tasks, etc.). The proposed framework for rate analysis can be used for fast design space exploration to determine how these bounds change with different architectural parameters, mapping of the application, or changing the QoS requirements associated with the input streams.

1 Introduction

In recent years, there has been a considerable interest in design methodologies targeted towards developing multiprocessor System-on-Chip (SoC) platforms specifically for implementing multimedia applications (e.g. the Viper SoC architecture [5] from Philips). Many of these platforms are typically designed to process concurrent streams of audio and video data associated with broadband multimedia services and, at the same time, perform network packet processing to support high-speed Internet access.

Following the increasing need for design methodologies for such multiprocessor platforms, in this paper we address a problem which we refer to as the *rate analysis problem*. Given a multiprocessor architecture and a multimedia application that has been partitioned and mapped onto it, the problem we aim at is to determine *tight bounds* on the rates at which different multimedia streams can be fed into this architecture. This is an important issue since when a stream arrives at a rate that is higher than a certain upper bound, this may lead to buffer overflows in the architecture. This problem is especially acute when dealing with architectures for portable devices (such as PDAs and portable audio/video players) which have a very limited on-chip buffer memory. On the other hand, when the stream arrives at a rate which is lower than a specified threshold, the quality

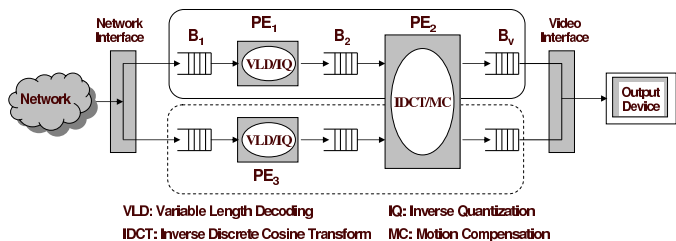


Figure 1. A multiprocessor SoC platform processing two concurrent MPEG-2 streams for a picture-in-picture (PiP) application.

of the output might suffer as well, i.e. the quality-of-service (QoS) constraints associated with the application may be violated. The goal of our rate analysis is to precisely compute these upper and lower bounds; this can help in designing wireless interfaces and suitable buffering and traffic shaping mechanisms for multimedia streams.

The main difficulties associated with the rate analysis problem stem from (i) the high data-dependent variability in the execution time of multimedia tasks [15], (ii) the burstiness in on-chip traffic arising from multimedia processing on multiprocessor platforms [17], and (iii) the presence of on-chip buffers and different scheduling algorithms implemented on the different architectural components. As a result, the rate analysis problem should *not* be restricted to computing a constant “long-term” arrival rate of a multimedia stream. It should rather be concerned with computing the allowable burstiness of a stream at different time scales. To address this issue, we use a very general event model to capture the burstiness in the arrival rates of multimedia streams. The generality of this event model makes the rate analysis problem non-trivial, as will become clear in the subsequent sections of this paper.

Given a multiprocessor platform architecture (such as the one shown in Figure 1), the bounds returned by rate analysis depend on architectural parameters such as the amount of on-chip memory available, clock frequencies of different processors and bus arbitration policies. Further, these bounds also depend on application characteristics, including how the application is partitioned and mapped onto the

architecture. If the input stream rates are dictated by the environment, a designer has to tune the platform configuration such that these rates can be supported by the architecture. The rate analysis framework that we present in this paper can help a system designer to precisely solve this problem and also identify all the tradeoffs involved.

Although *scheduling* of multimedia streams has been extensively studied in both, the multimedia and the real-time systems literature, the rate analysis problem has not been addressed in sufficient detail. Restricted versions of this problem have however been studied before in the embedded systems literature (e.g. computing maximum execution rates of concurrent processes interacting through synchronization). We believe that our work can also generalize some of these previous results. In addition, it can lead to new insights into less-studied problems like sensitivity of schedulability analysis on input parameters, and eventually also inspire new techniques for solving them.

The rest of this paper is organized as follows. In the next section we formally state our problem and introduce the necessary mathematical tools. In Section 3 we present our rate analysis framework. This is followed by our experimental results in Section 4. We discuss some related work in Section 5 and finally conclude in Section 6 by outlining some directions for future work.

2 Problem Formulation

Our system-level view of a multiprocessor media-processing platform is shown in Figure 1. This figure illustrates a picture-in-picture (PiP) application where two concurrent video streams are being processed by the platform architecture. An MPEG-2 decoder application is partitioned and mapped onto three processing elements (PEs) PE_1 , PE_2 and PE_3 . The VLD and IQ tasks of the decoder application have been mapped onto PE_1 and also replicated on PE_3 . Each of these two PEs process a different stream. PE_2 , on the other hand, implements the IDCT and MC tasks and processes both the streams. A scheduler implemented on PE_2 schedules these streams, probably using different QoS parameters for each stream. The stream corresponding to the main window in the PiP application might be associated with a higher frame rate and resolution and will generate a higher workload on PE_2 , compared to the stream associated with the secondary window. When a user switches off the PiP mode, the stream associated with the secondary window is switched off and all the processor cycles in PE_2 are used for the main stream.

As shown in Figure 1, the video streams arrive over a network and enter the platform after some initial packet processing at the network interface. These streams are stored in an on-chip buffer memory, such as B_1 . PE_1 reads a bitstream from the buffer B_1 , processes it and the partially processed stream (consisting of partially decoded

macroblocks) is written into the buffer B_2 . B_2 is a FIFO channel which is read by PE_2 for further processing of the stream. Finally, the fully decoded macroblocks are written into the playout buffer B_v which is read by the output video device according to some pre-specified rate.

Typical design constraints that need to be satisfied in a setup like this are (i) the playout buffers should not underflow (this would result in the output device missing a frame to be displayed), and (ii) none of the buffers should overflow. Note that because of factors like congestion in the network, the bitstream arriving at the buffer B_1 might be bursty in nature. However, the amount of burstiness would also depend on the kind of processing and buffering done at the network interface. In addition to this, the number of bits consumed by the VLD/IQ task to produce one partially decoded macroblock at the output of PE_1 is also highly variable. Lastly, the number of processor cycles required in this process (i.e. to generate one partially decoded macroblock) is also variable. For many multimedia tasks, the ratio between the worst-case and the average load on the processor can be as high as a factor of 10 [15]. As a result, the stream of partially decoded macroblocks that get written into B_2 will be highly bursty in nature.

To simplify the ensuing discussion, we shall consider a multimedia stream to be made up of a potentially infinite sequence of *stream objects*. Such a stream object might be a bit, a macroblock, or a video frame depending on which processing stage the stream is in. For example, at the input to the network interface a stream object can be a network packet, in the buffer B_1 a stream object is a bit and in the buffer B_2 it is a partially processed macroblock.

Given a scheduling policy (and its associated parameters) for PE_2 , the sizes of the buffers B_2 and B_v , and the rate at which B_v is read out by the output device, we want to compute tight bounds on the rate at which the stream objects can be allowed to arrive at B_2 , such that the buffer overflow and underflow constraints are satisfied.

Recall from the above discussion that the rate at which stream objects are written out by PE_1 into B_2 is highly bursty in nature. If this rate “matches” the bounds mentioned above, then all buffer constraints will be satisfied for the system. However, if these rates do not match, then certain parts of the architecture will have to be tuned accordingly. This tuning might include (i) changing buffer sizes, (ii) changing the scheduler in PE_2 , (iii) changing the task mapping, or (iv) modifying the network interface. On the other hand, if the bounds allowed by PE_2 are much larger than those at which PE_1 outputs stream objects, then certain buffer sizes can be reduced to save cost.

Since we are concerned with the *rates of bursty* streams, it is not sufficient to specify such rates solely using the “long-term” arrival rates of stream objects. We would rather want to accurately specify the amount of burstiness in a

stream. Towards this end, we shall use the concept of *variability characterization curves* (VCC) which was recently introduced in [12] by extending results from the theory of *Network Calculus* [3]. Here we briefly discuss the basics of VCCs and then show how they can be used to derive bounds on the burstiness of a stream. We will also use VCCs to capture the data-dependent *variability* in the execution requirements of stream objects and the *service* offered by a processor to a stream.

2.1 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. For any reasonably large value of k , $\gamma^l(k)$ will most likely be greater than $k \times e_{\min}$ (since *all* the k stream objects are unlikely have the minimum possible execution requirement e_{\min}). Further, the difference between them increases with increasing values of k . Similarly, $\gamma^u(k)$ will be smaller than $k \times e_{\max}$ for the same reason. Hence, the VCC γ is more expressive compared to a straightforward best- or worst-case characterization of the execution requirements of sequences.

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

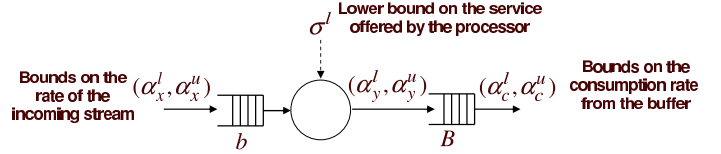


Figure 2. Processing a single stream.

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 1), $\alpha^l(\Delta)$ and $\alpha^u(\Delta)$ denote the minimum and the maximum number of stream objects that arrive within *any* time interval of length Δ . Figure 2 shows a single stream that is being processed by a PE. Here, (α_x^l, α_x^u) are used to represent the incoming stream, (α_y^l, α_y^u) represent the processed stream and (α_c^l, α_c^u) represent the bounds on the rate at which the stream is consumed from the buffer B . We will often refer to (α_c^l, α_c^u) as the *consumption bounds*.

As an example, let $\alpha_x^l(10) = \alpha_x^u(10) = 5$, which essentially means that within any time interval of length 10, at least and at most 5 stream objects can arrive at buffer b . Hence, the *average* arrival rate is one stream object in every two time units. Now suppose that we are also given that $\alpha_x^u(2) = 4$, which means that within a time interval of length 2 there might be a burst of at most 4 stream objects. Following this specification, if 4 stream objects arrive at b during the time interval $[0, 2]$, then over the time interval $(2, 10]$ at most 1 stream object can arrive. Hence, although the “long-term/average” arrival rate of the stream is 0.5 stream objects per unit time, there might be occasional bursts. The arrival curves α^l and α^u allow for a precise characterization of such bursts.

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 serviced) by a processor within *any* time interval of length Δ . β^l and β^u (as well as γ^l and γ^u) may be derived from a trace of execution requirements of stream objects obtained from a set of *representative* audio/video clips.

Note that this specification of *service* is stream dependent. It is also possible to specify the service offered by a processor in a stream-independent manner. Towards this, let $\sigma^l(\Delta)$ and $\sigma^u(\Delta)$ denote the minimum and the maximum number of processor cycles available within any time

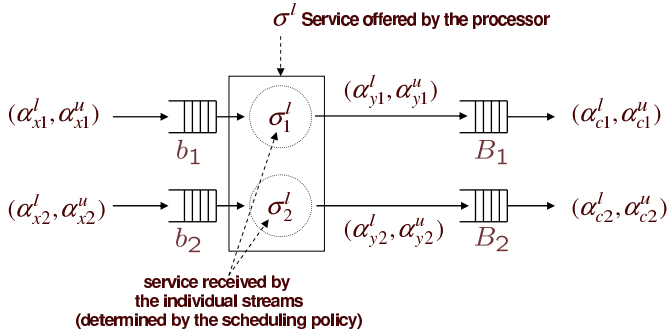


Figure 3. Processing multiple streams.

interval of length Δ . It is then easy to see that $\beta^l(\Delta) = \gamma^{u-1}(\sigma^l(\Delta))$ where γ^u is the workload curve associated with the stream (or a set of representative streams).

2.2 The Rate Analysis Problem

We are now ready to formally state the rate analysis problem. Let us again consider Figure 2. Suppose that we are given α_c^l and α_c^u , a lower bound or guaranteed service offered by the PE (i.e. σ^l), the workload curves γ^l and γ^u and the buffer sizes b and B . The rate analysis problem is then to compute the functions α_x^l and α_x^u such that the buffer B does not underflow and neither of the buffers (i.e. b and B) overflow. It can then be guaranteed that any stream whose arrival process is bounded by α_x^l and α_x^u will satisfy the buffer underflow and overflow constraints.

Note that the VCCs always capture the characteristics (e.g. arrival pattern, execution demand, etc.) of a *class* or *set* of streams. For example, the workload curves γ^l and γ^u capture *all* possible execution traces for which *any* k consecutive stream objects require a minimum of $\gamma^l(k)$ and a maximum of $\gamma^u(k)$ processor cycles. Hence, the problem specification given above holds for not just one concrete stream, but a class of streams. An example of such a class might be all video clips having the same resolution and bitrate. As mentioned in the previous subsection, such VCCs may be obtained from a few audio/video clips that are representative of the class we would like to capture (see [11] for details).

When multiple streams are being processed by a PE, as shown in Figure 3, we are also given a specification of the scheduler running on the PE. The problem in this case is to compute the functions α_x^l and α_x^u for each of the individual streams. Again, the computed arrival curves are required to satisfy the buffer overflow and underflow constraints.

Finally, in the case of architectures with multiple pipelined PEs, the challenge is to propagate the results of rate analysis from one PE to the next, starting from the one closest to the output device (which is PE_2 in Figure 1). The final result of such an analysis will then be precise bounds on the input rate at which a stream can be fed into the platform architecture.

3 Rate Analysis

In this section, we first present our rate analysis framework for the case of a single PE that is closest to the output device. We will then show how to extend it to the case of other PEs in the path of a stream. To proceed, we first need to introduce some notation and a technical result (see [3] for additional background).

Notation. Throughout this paper, all functions f are assumed to be wide-sense increasing, meaning that $f(x_1) \leq f(x_2)$ for $x_1 \leq x_2$ and $f(x) = 0$ for $x \leq 0$. For any two functions f and g , the *min-plus convolution* of f and g is denoted by

$$(f \otimes g)(t) = \inf_{0 \leq s \leq t} \{f(t-s) + g(s)\}$$

and the *min-plus deconvolution* of f and g is denoted by

$$(f \oslash g)(t) = \sup_{u \geq 0} \{f(t+u) - g(u)\}$$

We will use $f \wedge g$ to denote the infimum or minimum (if it exists) of f and g , and $f \vee g$ to denote the supremum or maximum (if it exists) of f and g .

Lemma 1 For any three functions f , g and h , $g \otimes h \geq f$ if and only if $h \geq f \oslash g$.

This lemma follows from the definitions of the min-plus convolution and deconvolution operations and shows the relation between them.

3.1 The Single Stream Case

Let us again consider Figure 2. Using results from [3], it can be shown that the maximum backlog at the input buffer b is bounded by

$$\sup_{\Delta \geq 0} \{\alpha_x^u(\Delta) - \beta^l(\Delta)\} \quad (2)$$

where β^l can be obtained from γ^u and σ^l as discussed in Section 2.1. For the sake of notational simplicity, from here on we will use b and B to denote both, the buffers and their respective sizes.

Using this result, the constraint that the buffer b never overflows can be stated as:

$$\alpha_x^u(\Delta) \leq \beta^l(\Delta) + b, \quad \forall \Delta \geq 0 \quad (3)$$

Similarly, the constraint that the playout buffer never overflows can be stated as:

$$\alpha_y^u(\Delta) \leq \alpha_c^l(\Delta) + B, \quad \forall \Delta \geq 0 \quad (4)$$

Let the playback delay associated with the output device be equal to t_d , i.e. the first stream object is read out from the playout buffer B at time $t = t_d$. Thereafter, B is read out at a rate specified by the consumption bounds α_c^l and α_c^u . The time interval $t = [0, t_d)$, is often referred to as the *buffering time*. We assume that the bounds (α_c^l, α_c^u) hold over the time interval $[t_d, \infty)$ i.e. the buffering time is ignored. This is needed in order to obtain tighter bounds.

Ineq. (4) guarantees that the buffer B never overflows subject to the condition that it is empty at the time t_d and

starts filling up from thereon. In reality, this is of course not true since stream objects are written into B during the buffering time. As a result, even if Ineq. (4) is satisfied, certain stream objects might be dropped. However, the maximum number of dropped stream objects can be bounded and we shall derive this bound at the end of this subsection.

It can also be shown that $\alpha_x^u(\Delta) = (\alpha_x^u \circ \beta^l)(\Delta)$. Using this result, Ineq. (4) is equivalent to

$$(\alpha_x^u \circ \beta^l)(\Delta) \leq \alpha_c^l(\Delta) + B, \quad \forall \Delta \geq 0$$

Using Lemma 1, this inequality can now be reformulated as:

$$\alpha_x^u(\Delta) \leq (\beta^l \otimes \alpha_c^l)(\Delta) + B, \quad \forall \Delta \geq 0 \quad (5)$$

By combining the Ineqs. (3) and (5), we obtain the following upper bound on α_x^u :

$$\alpha_x^u(\Delta) \leq (\beta^l(\Delta) + b) \wedge ((\beta^l \otimes \alpha_c^l)(\Delta) + B), \quad \forall \Delta \geq 0 \quad (6)$$

Next, we derive a lower bound on α_x^l . It may be noted that this bound will depend on the upper consumption bound α_c^u , the service curve β^l and the playback delay t_d . Let us first consider the case where the playback delay $t_d = 0$. In this case, the output device has to wait for a maximum of $\alpha_y^{l-1}(k) - \alpha_c^{u-1}(k)$ time units until the k -th stream object is written into the playout buffer B . From this it is possible to bound the maximum time interval for which the output device might have to wait to read a stream object. This bound is equal to: $\sup_{k \geq 0} \{\alpha_y^{l-1}(k) - \alpha_c^{u-1}(k)\}$

Using this result, we have the following theorem for non-zero t_d .

Theorem 1 *Given the upper consumption bound α_c^u , the lower output bound α_y^l , and the playback delay t_d , the playout buffer B will never underflow if*

$$\sup_{k \geq 0} \{\alpha_y^{l-1}(k) - \alpha_c^{u-1}(k)\} \leq t_d$$

Proof: Let $C(t)$ denote the number of stream objects consumed by the output device and $y(t)$ denote the number of stream objects processed by the PE during the interval $[0, t]$. When the inequality $\sup_{k \geq 0} \{\alpha_y^{l-1}(k) - \alpha_c^{u-1}(k)\} \leq t_d$ holds, it follows that

$$C(t) \leq \alpha_c^u(t - t_d) \leq \alpha_y^l(t) \leq y(t)$$

is also true. The guarantee that the playout buffer never underflows follows from the inequality $C(t) \leq y(t)$. \square

From this theorem it follows that in order for the playout buffer not to underflow, the following constraint needs to be satisfied:

$$\alpha_y^{l-1}(k) \leq \alpha_c^{u-1}(k) + t_d, \quad \forall k \geq 0 \quad (7)$$

For notational simplicity, we will use $\lambda_c^u(\Delta)$ to denote the *pseudo-inverse* (see Section 2.1) of the function $\alpha_c^{u-1}(k) + t_d$. Ineq. (7) can then be written as: $\alpha_y^l(\Delta) \geq \lambda_c^u(\Delta)$, $\forall \Delta \geq 0$. Further, it can be shown that $\alpha_y^l(\Delta) =$

$(\alpha_x^l \otimes \beta^l)(\Delta)$. By combining this with the above inequality, we obtain that $(\alpha_x^l \otimes \beta^l)(\Delta) \geq \lambda_c^u(\Delta)$. From Lemma 1 we can then obtain the following lower bound on α_x^l :

$$\alpha_x^l(\Delta) \geq (\lambda_c^u \circ \beta^l)(\Delta), \quad \forall \Delta \geq 0 \quad (8)$$

Inequalities (6) and (8) therefore give upper and lower bounds on the rate of the input stream. The functions $\alpha_x^l(\Delta)$ and $\alpha_x^u(\Delta)$ are hence solutions to our rate analysis problem – they not only specify the allowed long-term arrival rate of the stream, but also accurately characterize the maximum burstiness that can be tolerated.

Bounding the buffer overflow: Ineq. (4) guarantees that the playout buffer never overflows during the time interval $[t_d, \infty)$ subject to the condition that it is empty during $[0, t_d]$. In reality the assumption that B is empty during $[0, t_d]$ does not hold. However, it is possible to obtain an upper bound on the number of stream objects that can arrive within this time interval. This upper bound is the maximum number of stream objects that can overflow from the buffer as a result of the above assumption. We know that the maximum number of stream objects that can be processed within $[0, t_d]$ is $\gamma^{l-1}(\sigma^l(t_d))$, and the maximum number of stream objects that can arrive at b within $[0, t_d]$ is $\alpha_x^u(t_d)$. Hence, the maximum number of stream objects that can arrive at B during $[0, t_d]$ is $\min\{\gamma^{l-1}(\sigma^l(t_d)), \alpha_x^u(t_d)\}$, which is therefore equal to the maximum number of stream objects that can overflow from B .

3.2 The Case of Multiple Streams

Now let us consider the case where multiple streams are being processed by a PE. An example of this is PE_2 in Figure 1, which processes two streams. The problem specification is similar to the single stream case and is shown in Figure 3. Let us again assume that the PE of interest is the one next to the output device. Given are the consumption rates of the two streams by their respective output devices, the sizes of the playout and input buffers, the *total* service offered by the PE, and the scheduling policy implemented on the PE. The problem is again that of computing bounds on the input rates such that buffer overflow and underflow constraints are satisfied. In this case, if σ^l is the *total* service offered by the PE, then the scheduler implemented on this PE divides this service σ^l between the two streams (see Figure 3). The bounds on the service received by the two streams depend on the scheduling policy. We have computed these bounds for fixed priority and time division multiplexing scheduling policies. The same techniques may be applied to other scheduling disciplines as well. Once such bounds on the service received by each of the two streams have been computed, the rate analysis problem boils down to applying the analysis techniques developed for the single stream case to the individual streams. Due to space restrictions, we omit the details here; they may be found in [7].

3.3 Multiple Processing Elements

Our view of multimedia processing on a multiprocessor System-on-Chip platform, as outlined in Section 2, consists of multiple PEs processing any stream in a pipelined fashion. Between any two PEs a FIFO buffer stores the partially processed stream. The last PE in the path of a stream writes out the fully processed stream into the playout buffer, which is read out by an output device. The derivation of the bounds on the arrival rate of a stream, that we presented so far, was only concerned with this last PE, which feeds the playout buffer. Recall that the computed bounds pertain to the maximum and minimum rates at which a stream can arrive at the (input) buffer at the input to this PE. The constraints that the computed bounds were required to follow were (i) the playout buffer should not underflow, and (ii) none of the buffers should overflow. Among the inputs to our rate analysis problem were bounds on the consumption rate by the output device from the playout buffer, specified as upper and lower arrival curves (the consumption bounds (α_c^l, α_c^u)).

Let us now consider the PE (e.g. PE_1 in Figure 1) adjacent to this last PE in the path of the stream. To compute the bounds on the arrival rate of a stream at this PE, the input bounds computed for the downstream PE (i.e. PE_2 in Figure 1) serve as output bounds for this PE (i.e. the processed stream coming out of this PE must satisfy these bounds). The only buffer constraint that needs to be satisfied in this case is that the buffer at the input of this PE (B_1 in Figure 1) should not overflow. Deriving the input bounds on the arrival is therefore much more simpler than the case we considered above. This is because, only the following two constraints need to be satisfied: (i) Ineq. (3), and (ii) the bounds on the processed stream must be constrained by the input bounds computed for the adjacent downstream PE.

This process of computing the bounds on the arrival rate of a stream is cascaded to all the upstream PEs, until the first PE in the path of a stream is encountered. The input bounds computed for this PE therefore serve as bounds on the arrival rate of a stream entering the platform architecture.

3.4 Summary of our Contributions

In contrast to scheduling problems typically studied in the real-time systems literature, our framework has three novel features. (i) Rather than explicit deadline constraints, we dealt with buffer overflow and underflow constraints, which are often more relevant in the context of streaming applications. (ii) The event and service models we used were more expressive than traditionally studied models like periodic, sporadic, etc., and they can be used to precisely capture bursts and variabilities in the execution demands of streams. However, the resulting analysis became substantially more involved. (iii) The framework is fully composi-

tional and the analysis does not blowup when dealing with multiple PEs and streams. This framework is based on *deterministic queuing theory* developed in the context of communication networks [3], but was applied here to analyze real-time embedded systems.

4 Simulation Results

We validated our analytical framework using several detailed simulations. Towards this end, we implemented a transaction-level model of the platform architecture shown in Figure 1 using SystemC [16]. The on-chip PEs were modeled using the SimpleScalar instruction set simulator [1], in which we used the *sim-profile* configuration and the PISA instruction set.

We modeled each video stream at the macroblock granularity. For any given video clip, we first simulated its execution (decoding using an MPEG-2 decoder application) using SimpleScalar and obtained execution time traces of the VLD, IQ, IDCT and MC tasks. These traces record the execution requirement (in processor cycles) of each macroblock belonging to the video clip, for each of the above tasks. Based on these traces and the constant bitrate at which the video clip (which is a compressed bitstream) is fed into PE_1 or PE_3 (in Figure 1), it is possible to determine the arrival pattern of the stream at the input of PE_2 (e.g. at the buffer B_2) and also at the playout buffer (i.e. buffer B_v). For this, we used the SystemC-based transaction-level model of the architecture, which was also used to model the scheduling policy on PE_2 when multiple streams (two in this case) are processed by the architecture. From SystemC simulations, we measured the fill levels of the different buffers for any given video clip(s).

To validate our framework, we first compute bounds on the arrival rates of streams at the input of PE_2 (i.e. at B_2). We then show using simulation that video clips which respect these bounds also satisfy the buffer overflow and underflow constraints. At the same time, clips which do not respect the computed bounds, either result in buffer overflow or underflow, thereby showing that the computed bounds are not overly pessimistic. These bounds are non-trivial, in the sense that they precisely capture the allowed burstiness in a stream. Obtaining them using purely simulation-based techniques is clearly not possible due to the extensive simulation times involved. As discussed in Section 1, these bounds can provide useful insights for tuning the platform architecture (e.g. determining the optimal clock frequency of PE_1 and also designing the input network interface).

Recall that one of the inputs to our analytical framework, is the workload curve $\gamma(k)$ specifying lower and upper bounds on the number of processor cycles required by any k consecutive stream objects. For the bounds on the arrival rate of a stream—that are computed by our framework—to be useful, they should hold good for a *class* or *set* of

streams or video clips, and not for just a single video clip. As mentioned before, an example of such a *class* might be all video clips having the same bitrate and frame resolution. Therefore, the workload curve $\gamma(k)$ that we use as an input, should also specify the workload demand of the class of video clips we are interested in.

For our PiP application (see Section 2) that we used in our experiments, we chose two classes of video clips — those that have high motion content and second being made up of still images. The former class of clips are to be displayed in the main window of the PiP application; they are representative of usual video clips like movies. The latter class is representative of text messages or similar information about the main window being displayed in the secondary window of the PiP application. In what follows, for ease of exposition, we drop the term *class* when we talk about bounds on arrival rates; these bounds are always expected to hold for a class of streams and not just a single stream.

To obtain the workload curves corresponding to the above two classes, we simulated the execution of a set of MPEG-2 video clips using SimpleScalar, as mentioned above. We then analyzed the resulting execution time traces for the IDCT and MC tasks (which are mapped onto PE_2) and derived the bounds γ^l and γ^u (see Section 2). This procedure follows a recently developed technique described in [11]; we refer the interested reader to this paper for further details. The video clips for both these classes were encoded using a constant bitrate of 8 Mbps; they had a frame resolution of 704×576 pixels and a playback rate of 25 frames per second. Typically, the video clips displayed in the secondary window of a PiP application would have a lower resolution and bitrate than those displayed in the main window. However, for simplicity reasons, we decided to distinguish between the two classes only on the basis of their content (i.e. motion versus still videos).

For reporting our experimental results, we denote the computed arrival rates of a stream at the input of PE_2 using (α_x^l, α_x^u) . To validate these bounds, we compared them with similar bounds obtained from simulation using individual video clips. We denote these bounds as (α_m^l, α_m^u) (where the subscript m denotes “measured”). Towards this end, we first record the trace of arrival times of partially decoded macroblocks at the input of PE_2 and then analyze these traces to obtain the bounds (α_m^l, α_m^u) (exactly as the workload curves γ^l and γ^u were derived). To measure the fill levels of buffers, if B_s is the specified buffer size and B_d is the computed upper bound on the number of stream objects that might be dropped then $B_s + B_d$ is an upper bound on the maximum buffer fill level, which we want to validate using simulations. Similarly, the fill level of a playout buffer should always be greater than 0 in order to satisfy the underflow constraint.

class	scenario	i/b size (mb)	p/b size (mb)	video clip	violation ?
motion	1	4000	5600	A	no
	2	4000	5600	B	yes
	3	3500	8000	C	no
still	4	5000	3000	D	yes
	5	4000	5600	E	no
	6	3000	3000	F	no

i/b: input buffer; **p/b:** playout buffer (sizes in macroblocks)

Table 1. Scenarios for the single stream case. Video clips from: ftp.tek.com/tv/test/streams/Element/MPEG-Video/

For our experiments, we used a selection of *scenarios* shown in Table 1. Each scenario is specified by a class (of video clips), the input and playout buffer sizes and a video clip belonging to the class. The bounds on the arrival rates are computed from the class information and the buffer sizes. These are compared with the simulation results based on the buffer sizes and a concrete clip belonging to the class. For all the experiments, we run PE_2 at a constant frequency. Hence, the service offered by it can be represented as $\sigma^l(\Delta) = c \cdot \Delta$, where c is the frequency.

In the following section, we show our results for the single stream case. The results for the case of multiple streams are described in [7].

4.1 The Single Stream Case

In this case, the PiP mode is switched off. For clarity of presentation, instead of plotting the functions α_x^l, α_x^u , etc. directly, we plot the differences $\alpha_x^u - \alpha_x^l, \alpha_m^u - \alpha_x^l$ and $\alpha_m^l - \alpha_x^l$. Clearly, the arrival process of a video clip, which is captured in (α_m^l, α_m^u) , violates the analytically computed bounds (α_x^l, α_x^u) whenever $\alpha_m^u - \alpha_x^l$ or $\alpha_m^l - \alpha_x^l$ crosses $\alpha_x^u - \alpha_x^l$ or goes below 0.

These plots are shown in Figure 4 for three different scenarios. The same figure also shows the fill levels of the input and the playout buffers (i.e. B_2 and B_v in Figure 1). Note that for Scenario 1, the measured arrival patterns satisfy the analytically computed bounds (Figure 4(a)). For this scenario, the measured fill levels of the input buffer (Figure 4(b)) and the playout buffer (Figure 4(c)) are less than the computed upper bounds. Also note that beyond the playback delay, the playout buffer does not underflow.

In Scenario 2, the measured upper arrival curve α_m^u violates the computed upper bound α_x^u . In this case, the measured buffer fill levels are greater than the sum of the specified buffer sizes and the upper bounds on the number of macroblocks that might be dropped. This is indicated as *buffer overflow* in Figure 4. Finally, in Scenario 4, the measured lower arrival curve α_m^l violates the computed lower bound α_x^l . In this case, the simulation results show that the

playout buffer underflows. For the remaining three scenarios (i.e. 3, 5 and 6), all the buffer constraints are satisfied, as indicated in Table 1. Hence, they are not shown in Figure 4. Note that for all the scenarios, the input buffer sometimes underflows. However, this does not affect the performance of the system and we also do not specify it as a constraint.

Figure 5 shows the computed bounds on the buffer fill levels and the measured fill levels obtained using simulation, for all the six scenarios. From Table 1, note that apart from Scenarios 2 and 4, the measured arrival bounds always satisfy the computed bounds. Figure 5 confirms that it is only for these two scenarios that buffers either overflow or underflow, thereby validating our proposed framework.

5 Related Work

The rate analysis problem has been studied before in the embedded systems domain, albeit in a different context. Broadly speaking, the setup considered before consists of a collection of concurrently executing embedded systems components or processes that interact through synchronization messages. The problem is to compute bounds on the execution rates of these processes, given certain resource constraints. Alternatively, given a number of rate constraints, the problem is to efficiently check if these constraints are consistent. Often, it is required to check these constraints in an interactive fashion and hence the emphasis in such cases has been on appropriate tool support. We refer the interested reader to [4, 9] and the references therein, for details on this line of work.

In this paper we were concerned with the rate analysis problem in the context of processing multiple concurrent multimedia streams. Rather than computing bounds on the execution rates of a process, as in [4, 9], our aim has been to compute the allowable bursts in a multimedia stream over different time scales. Such bursts are specified as *arrival curves* which bound the minimum and maximum number of data items or events that can arrive at the system within any specified time interval length. We believe that our results can be combined with the work in [4, 9] to model and analyze reactive systems consisting of a number of interacting processes that are triggered by bursty event streams. More specifically, the work in [9] is only concerned with a periodic model, where the different interacting processes execute in a periodic fashion. As a first step, this restriction can be removed using our event model which allows the specification of arbitrary, but bounded bursts.

Within the real-time systems area, there has been a growing interest in the problem of computing the *parameter space* for which a system becomes schedulable. A recent paper [6] addressed the problem of computing the end-to-end feasibility regions of distributed aperiodic task systems under fixed-priority scheduling. The goal here was to compute the multidimensional space—with each dimension

as the utilization of a resource—within which the system meets certain end-to-end deadlines. Similarly, [2] addressed the problem of identifying task activation rates for fixed-priority scheduled systems that meet certain deadline constraints. The work that we presented here is in the same general direction as that of the abovementioned two papers.

Our work has been inspired by a recent paper [10] which studied the rate analysis problem for multimedia streams. However, in contrast to our work, this paper computes the bounds on the arrival pattern of an input stream using two functions $x_{min}(t)$ and $x_{max}(t)$. Any arrival pattern $x(t)$, which is bounded by these two functions, i.e. $x_{min}(t) \leq x(t) \leq x_{max}(t)$, is guaranteed to satisfy all buffer overflow and underflow constraints (exactly as we specify here). The function $x(t)$ denotes the number of stream objects that can arrive at the system during the time interval $[0, t]$. The use of such a *concrete* arrival trace—rather than *bounds* on the burstiness, as we do here—considerably simplifies the formulation of the buffer underflow and overflow constraints. The downside of such a formulation is that the resulting bounds (x_{min} and x_{max}) are considerably more pessimistic than the bounds we derive in the present paper.

To see this, we have used Scenario 1 in Table 1 to analytically compute the bounds x_{min} and x_{max} based on the framework presented in [10]. These bounds are compared with the bounds α_x^l and α_x^u (that we obtained in this paper) in Figure 6. Figure 6(a) shows these bounds with the playback delay set to 0.3 sec, while Figure 6(b) was obtained with the playback delay set to 1.0 sec. It follows from our framework that for any t , the value of $x(t)$ can be as large as $\alpha_x^u(t)$. On the other hand, $x(t)$ can only be as large as $x_{max}(t)$ if the bounds derived in [10] are to be used. The reason behind the bounds x_{min} and x_{max} being pessimistic is that these bounds do *not* capture the burstiness in a stream. Given a concrete arrival pattern $x(t)$, which is bounded by x_{min} and x_{max} , let α_x^l and α_x^u denote the arrival curves which bound $x(t)$. It is very likely that $\alpha_x^u(t)$ will be greater than $x_{max}(t)$, especially for small values of t . Similarly, $\alpha_x^l(t)$ is likely to be less than $x_{min}(t)$.

Apart from the fact that the formulation of the buffer overflow and underflow constraints are more difficult in the case we consider in this paper, we also exploit the variability in the execution requirements of a stream (captured using the workload curves). This variability is not exploited in [10]. However, the bounds shown in Figure 6 do not make use of the workload curves when deriving α_x^l and α_x^u . These bounds were computed with the service curve $\beta^l(\Delta)$ set to $C(\Delta)$ (the constant consumption rate of the stream from the playout buffer). The bounds x_{min} and x_{max} were also computed with the same $\beta^l(\Delta)$. This was done to achieve a fair comparison between the two frameworks. Figure 6 clearly shows that the bound $[x_{min}, x_{max}]$ is significantly more pessimistic than the bound $[\alpha_x^l, \alpha_x^u]$ we derived here.

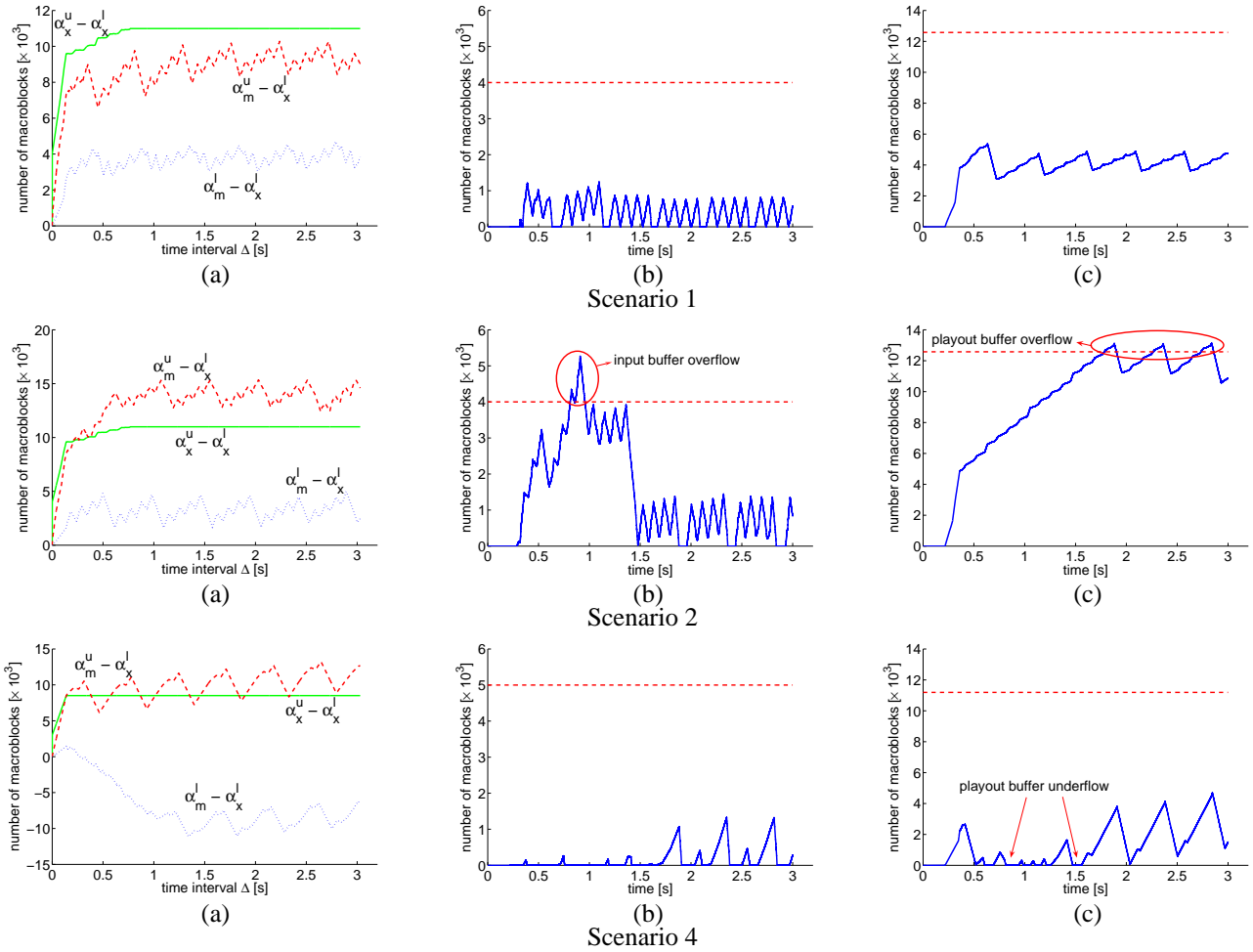


Figure 4. (a) Computed and measured bounds on the arrival rate, (b) Measured input buffer fill level, (c) Measured playout buffer fill level.

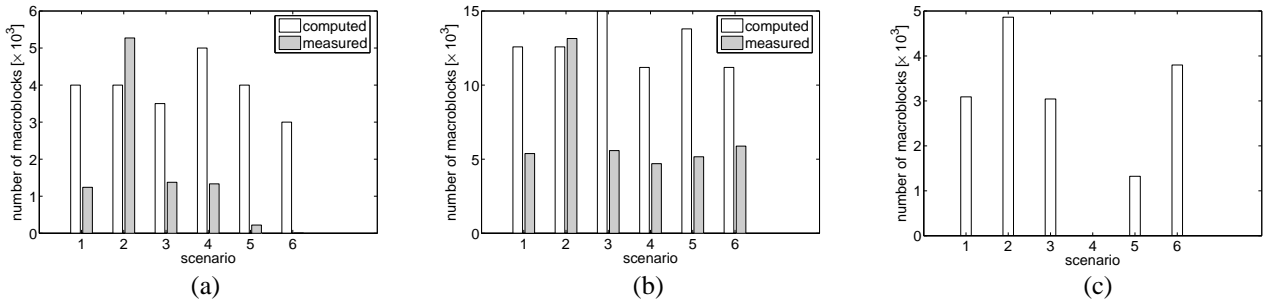


Figure 5. Buffer fill levels in the single stream case (a) Computed versus measured maximum fill level of the input buffer, (b) Computed versus measured maximum fill level of the playout buffer, (c) Measured minimum playout buffer fill level.

6 Concluding Remarks

In this paper we presented a framework for rate analysis for multiprocessor media-processing platforms. In contrast to [10]—which inspired our work—the bounds on the arrival rate of a stream that we derived precisely capture the allowed burstiness. This is especially important in the context of multimedia applications since they exhibit a high degree

of variability in their execution requirements. This variability becomes even more pronounced when such applications run on heterogeneous multiprocessor architectures implementing different scheduling and arbitration policies [13, 14].

All the results derived here were based on a deterministic *best/worst-case* analysis of multimedia streams and their processing requirements. Recently, a framework was

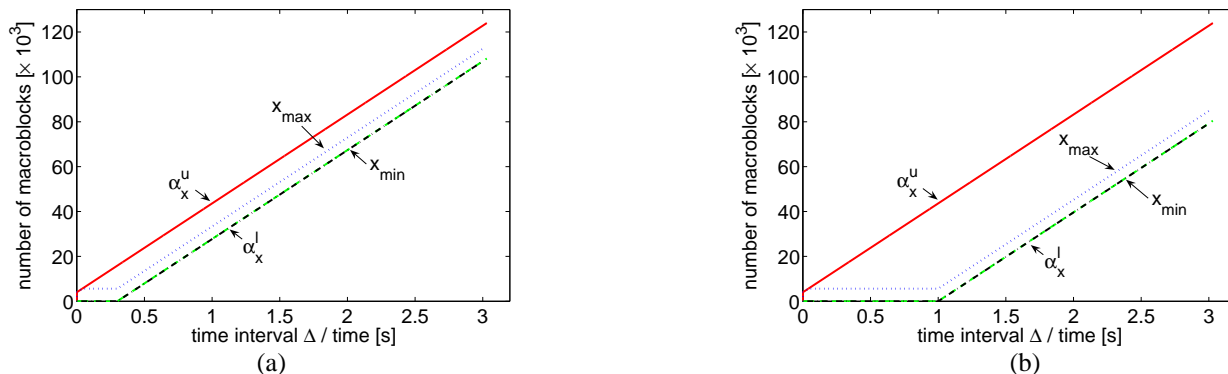


Figure 6. Bounds on the arrival rate of a stream (x_{min}, x_{max}) and (α_x^l, α_x^u) for two different playback delay values of (a) 0.3 sec and (b) 1.0 sec.

presented in [8] for *average*-case analysis of multimedia streams. The basic idea is that if a small deterioration in the output quality can be tolerated then this can often lead to significant resource savings. It would be interesting to solve the rate analysis problem using this framework.

Finally, as mentioned in Section 5, it would be worthwhile to explore possible combinations of the work in [4, 9] with our framework. More specifically, our application model can be extended to allow for arbitrary task graphs along with deadline constraints, in addition to the buffer constraints which we addressed here. We also believe that synergy between our work and those in [2, 6] would be interesting to explore as a part of future work.

Acknowledgements: This research was partially funded by the NUS URC grant R-252-000-190-112. The authors would also like to thank the anonymous reviewers for their suggestions, which helped in improving the paper.

References

- [1] T. Austin, E. Larson, and D. Ernst. SimpleScalar: An infrastructure for computer system modeling. *IEEE Computer*, 35(2):59–67, 2002.
- [2] E. Bini and M. D. Natale. Optimal task rate selection in fixed priority systems. In *IEEE Real-time Systems Symposium (RTSS)*, 2005.
- [3] J.-Y. L. Boudec and P. Thiran. *Network Calculus - A Theory of Deterministic Queuing Systems for the Internet*. Springer-Verlag LNCS 2050, 2001.
- [4] A. Dasdan, D. Ramanathan, and R. K. Gupta. A time-driven design and validation methodology for embedded real-time systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 3(4):533–553, 1998.
- [5] S. Dutta, R. Jensen, and A. Rieckmann. Viper: A multiprocessor SOC for advanced set-top box and digital TV systems. *IEEE Design & Test of Computers*, 18(5):21–31, September-October 2001.
- [6] W. Hawkins and T. Abdelzaher. Towards feasible region calculus: An end-to-end schedulability analysis of real-time multistage execution. In *IEEE Real-time Systems Symposium (RTSS)*, 2005.
- [7] Y. Liu, S. Chakraborty, and R. Marculescu. Generalized rate analysis for media processing platforms. http://www.comp.nus.edu.sg/~samarjit/rate_analysisTR.pdf.
- [8] Y. Liu, S. Chakraborty, and W. Ooi. Approximate VCCs: A new characterization of multimedia workloads for system-level MpSoC design. In *Design Automation Conference (DAC)*, 2005.
- [9] A. Mathur, A. Dasdan, and R. K. Gupta. Rate analysis for embedded systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 3(3):408–436, 1998.
- [10] A. Maxiaguine, S. Künzli, S. Chakraborty, and L. Thiele. Rate analysis for streaming applications with on-chip buffer constraints. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2004.
- [11] A. Maxiaguine, Y. Liu, S. Chakraborty, and W. Ooi. Identifying “representative” workloads in designing MpSoC platforms for media processing. In *IEEE Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia)*, 2004.
- [12] A. Maxiaguine, Y. Zhu, S. Chakraborty, and W.-F. Wong. Tuning SoC platforms for multimedia processing: Identifying limits and tradeoffs. In *International Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS)*, 2004.
- [13] K. Richter, M. Jersak, and R. Ernst. A formal approach to MpSoC performance verification. *IEEE Computer*, 36(4):60–67, 2003.
- [14] K. Richter, D. Ziegenbein, M. Jersak, and R. Ernst. Model composition for scheduling analysis in platform design. In *Design Automation Conference (DAC)*, 2002.
- [15] M. Rutten, J. van Eijndhoven, and E.-J. Pol. Robust media processing in a flexible and cost-effective network of multitasking coprocessors. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2002.
- [16] Open SystemC Initiative. <http://www.systemc.org>.
- [17] G. Varatkar and R. Marculescu. On-chip traffic modeling and synthesis for MPEG-2 video applications. *IEEE Transactions on VLSI*, 12(1), January 2004.