# Rate Analysis for Streaming Applications with On-chip Buffer Constraints

Alexander Maxiaguine
ETH Zürich
maxiagui@tik.ee.ethz.ch

Simon Künzli
ETH Zürich
kuenzli@tik.ee.ethz.ch

Samarjit Chakraborty
National University of Singapore
samarjit@comp.nus.edu.sg

Lothar Thiele
ETH Zürich
thiele@tik.ee.ethz.ch

**Abstract— While mapping a streaming (such as multimedia or network packet processing) application onto a specified architecture, an important issue is to determine the input stream rates that can be supported by the architecture for any given mapping. This is subject to typical constraints such as on-chip buffers should not overflow, and specified *playout buffers* (which feed audio or video devices) should not underflow, so that the quality of the audio/video output is maintained. The main difficulty in this problem arises from the high variability in execution times of stream processing algorithms, coupled with the bursty nature of the streams to be processed. In this paper we present a mathematical framework for such a *rate analysis* for streaming applications, and illustrate its feasibility through a detailed case study of a MPEG-2 decoder application. When integrated into a tool for automated design-space exploration, such an analysis can be used for fast performance evaluation of different stream processing architectures.**

## I. Introduction

Lately, there has been a tremendous increase in portable and mobile devices running algorithms for processing streams of audio and video data, and sometimes network packets. These include hand-held computers and mobile phones, and it is expected that their usage will increase even more in the future. Such devices typically have very stringent constraints pertaining to cost, size, and power consumption, and have posed several challenges towards developing appropriate models, methodologies, languages and tools for designing them (for example, see [10, 19, 20]).

The architecture of such devices typically consists of multiple processing elements (PEs) onto which parts of an application are mapped, and they are integrated on a single chip following a system-on-a-chip (SoC) design paradigm. In this setup, a system-level view of stream processing is as follows: the input stream enters a PE, gets processed by a function or algorithm implemented on this PE, and the processed stream enters another PE for further processing. Between two such PEs there is a buffer which stores the intermediate stream. Finally, the fully processed stream emerges out of a PE and gets stored in a *playout buffer* which feeds some *real-time client* such as an audio or video output device. The process of mapping a stream processing application onto such a target architecture gives rise to the problem of determining the range of input stream rates that can be supported by the architecture for a given mapping. Any feasible implementation, or mapping, of an algorithm onto an architecture is subject to constraints such as (i) the buffers between any two PEs should not overflow, and (ii) the playout buffer, which is read by the real-time client at some specified rate (depending on the quality of the audio or video output required) should not underflow at any point in time. Determining the range of feasible input stream rates, subject to the above constraints is difficult because of two main reasons. Firstly, there is a high data-dependent variability in the execution time of many stream processing algorithms, because it depends on the properties of the particular audio/video sample being processed. Secondly, the input streams themselves tend to be bursty in nature. These two factors coupled together can result in increasing the burstiness of the stream coming out of a PE, thereby necessitating a large amount of on-chip buffer space for its storage. Here, it may be noted that in contrast to the simple setup described above, there might be multiple streams being processed by a PE, where the different streams are processed by different functions—all of which are implemented on the same PE. The burstiness of the outgoing processed streams in such cases would additionally depend on the scheduling policy used to schedule them on the PE [16].

The importance of the above problem of *rate analysis* stems from the fact that on-chip buffers are available only at a premium, because of their large area requirements (see [9]). Therefore, when mapping a streaming application onto a specified architecture, it is necessary to accurately identify the feasible range of input stream rates (and bursts) that can be supported by the available on-chip buffers. This also includes the minimum rate that should be maintained to ensure the quality of the audio/video output.

### A. Our results and relation to previous work

There is a large body of work on on-chip traffic analysis and SoC communication architecture design (see [13, 14] and the references therein) which is relevant to the problem addressed in this paper. However, most of this relies on simulation based approaches. In the context of our problem, it typically requires several hours to simulate a few minutes of audio/video data for any reasonably detailed processor model. Further, simulation based approaches often fail to accurately characterize the allowed input rates and burst ranges, and strongly depend on the audio or video data used, which itself is difficult to select.

In this paper we present a mathematical framework for rate analysis for streaming applications, with the aim of overcoming the main problems associated with simulation based methods. For the sake of generality, we consider any *stream* to be made up of a potentially infinite sequence of *stream objects*, where a stream object might be a macroblock, a video frame, an audio sample, or a network packet, depending on the application in question. Given a specification of the architecture along with the different buffer sizes, the scheduling policies

implemented at the different PEs, the execution requirement per stream object of each processing function implemented on a PE, and the output rate required to drive the real-time client (such as the audio or video terminal), the proposed framework can be used to compute the minimum and maximum rate of the input stream. Here, *rate* refers to the precise characterization of the stream—including the allowed burst range/jitter and the long-term arrival rate—which are described more precisely in Section II. Any input stream whose rate is in between the computed minimum and maximum rates is guaranteed to satisfy all the constraints pertaining to buffer overflow and underflow. We also substantiate these theoretical results through a detailed case study of mapping a MPEG-2 decoder application on a specified architecture.

The proposed framework can be used to drive a system-level design space exploration where different possible mappings of a streaming application onto an architecture with fixed buffers need to be evaluated. It can also be used for optimal *buffer sizing* in an architecture, which is of crucial importance due to the high space requirements of on-chip buffers.

Our framework is based on the theory of *Network Calculus* [4], which extends the concept of *service curves* proposed in [7, 8] by placing it in an algebraic setting developed in [2]. Although Network Calculus was originally developed, and is still being largely used in the domain of communication networks, very recently it was used to analyse SoC architectures in the context of network processors [6, 18]. This work was further extended in [5, 11]. Our work in this paper follows this line of development. On an abstract level, all the previous papers considered the problem: given an input stream and the scheduling policy at each PE, what is the worst-case buffer requirement and what is the nature of the output stream? However, the problem addressed in this paper is the "reverse problem", where the output stream and the buffer size are given and the nature of the input stream needs to be computed. It turns out that none of the previous results can be extended to address this question, and a more elaborate theory based on [2] is required. Our work is also related to the problem of multimedia smoothing in the domain of communication networks [4], which addresses the problems of *shaping* an input stream to meet buffer constraints and that of computing the optimal *playback delay* or *buffering time* to maintain quality of service. It may be noted here that the main differences between the domains of communication networks and on-chip communication in SoC architectures are that in the former case (i) buffers are more readily available since there are no space constraints, (ii) packet dropping is a feasible option, which might not be possible in the latter case due to power/performance constraints, and (iii) *shaping* can be employed to reduce buffer consumption, which might be too costly to employ in the latter case. Lastly, related to this paper is also the work in [20], which proposes that on-chip traffic for multimedia applications exhibits *self-similarity*, and uses this property for optimal buffer sizing.

The problem is formally defined in the next section, followed by the details of the proposed framework. In Section III we present the case study of mapping a MPEG-2 decoder on a specified architecture. Finally, in Section IV we outline some of the possible directions in which this work may be extended.

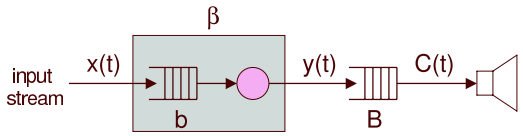


Fig. 1. A node with a processing element and an internal buffer of size $b$, processing an input stream and feeding the processed stream into a playout buffer of size $B$.

## II. Rate Analysis with Buffer Constraints

In this section we first state the problem definition, followed by some notation and then the case of a single PE with a playout buffer. This is then extended to consider the case of a stream passing through multiple PEs.

As mentioned in the last section, a stream is processed by multiple *nodes*, where each node consists of a PE and an internal buffer. Let us first consider the last node in the path of a stream, which feeds the processed stream into a playout buffer of size $B$, as shown in Figure 1. Let this node consist of a PE and an internal buffer of size $b$. The playout buffer is read by a real-time client such as an audio/video output device, at some specified rate. Let the input stream entering the node be denoted by $x(t)$, where $x(t)$ denotes the number of stream objects that arrived during the time interval $[0, t]$. The PE provides a guaranteed service $\beta(\Delta)$ of the following form: within *any* time interval of length $\Delta$, it will be able to process *at least* $\beta(\Delta)$ number of input stream objects. The function $\beta$ therefore provides a lower bound on the service provided by the PE, and is determined by the time required to process each stream object and the scheduling policy implemented at the PE (in case multiple streams or other tasks are also being processed by it). Let us denote the processed output stream entering the playout buffer by $y(t)$, which (like $x(t)$) denotes the number of stream objects coming out during the time interval $[0, t]$. The real-time client consumes stream objects from the playout buffer at a rate $C(t)$, which denotes the number of stream objects consumed within the time interval $[0, t]$.

Therefore, $x$, $y$ and $C$ are functions denoting *cumulative* values, while the function $\beta$ denotes values over *time interval lengths* and is referred to as a *service curve* [7]. Throughout this paper, we assume all functions $f$ to be wide-sense increasing (which means $f(s) \leq f(t),\ \forall s \leq t$), and $f(t) = 0$ for $t \leq 0$. Now, given $\beta$, $C$, and the buffer sizes $b$ and $B$, the problem is to compute the function (or set of possible functions) $x(t)$, such that (i) the playout buffer does not overflow, (ii) it does not underflow, and (iii) the internal buffer at the node does not overflow. These constraints are subject to the real-time server consuming stream objects at the specified rate $C(t)$ and the processing element providing a guaranteed service $\beta$. The version of the problem with multiple PEs is a simple extension of this, and is stated later.

As mentioned before, the constraint on playout buffer underflow is to maintain the quality of the audio/video output. The constraints on buffer overflow is motivated by the fact that typically static scheduling policies are implemented on the PEs (for simplicity), and hence checking buffer fill-levels and stalling a processor in case an output buffer is full, can not be easily implemented.

### A. Notation

For any two functions $f$ and $g$, the min-plus convolution of $f$ and $g$ is given by: $(f \otimes g)(t) = \inf_{s:0\leq s\leq t}\{f(t-s) + g(s)\}$. The min-plus deconvolution of $f$ and $g$ is given by: $(f \oslash g)(t) = \sup_{u:u\geq 0}\{f(t+u) - g(u)\}$. We use $f \wedge g$ to denote the infimum of $f$ and $g$, or the minimum if it exists, and $f \vee g$ to denote the supremum of $f$ and $g$, or the maximum if it exists.

### B. Buffer underflow and overflow constraints

Following our problem description, the constraint on the playout buffer underflow can be stated as:

$$y(t) \geq C(t) \quad \forall t \geq 0 \tag{1}$$

Since the PE provides a service guarantee of $\beta$, it can be shown that $y(t) \geq (x \otimes \beta)(t)$ (see [4] for details). Hence, the minimum value of $y(t)$ is equal to $(x \otimes \beta)(t)$ and the constraint given by Eqn.(1) can be reformulated as:

$$(x \otimes \beta)(t) \geq C(t) \quad \forall t \geq 0 \tag{2}$$

It can be shown that for any functions $f$, $g$ and $h$, $g \otimes h \geq f$ if and only if $h \geq f \oslash g$. Using this, Eqn.(2) can be stated as:

$$x(t) \geq (C \oslash \beta)(t) \quad \forall t \geq 0 \tag{3}$$

Similarly, the constraint on the playout buffer overflow can be stated as: $y(t) - C(t) \leq B \quad \forall t \geq 0$. Since $y(t)$ can be as large as $x(t)$ but not larger, this constraint can be reformulated as:

$$x(t) \leq C(t) + B \quad \forall t \geq 0 \tag{4}$$

Finally, the constraint that the internal buffer at the node should not overflow, is given by: $x(t) - y(t) \leq b \quad \forall t \geq 0$. Since $y(t) \geq (x \otimes \beta)(t)$, the minimum value of $y(t)$ is $(x \otimes \beta)(t)$ and the above constraint, as before, can be formulated as:

$$x(t) \leq (x \otimes \beta)(t) + b \quad \forall t \geq 0 \tag{5}$$

Eqns.(3), (4) and (5) therefore state all the constraints that the input stream $x(t)$ is required to satisfy.

### C. Computing bounds on $x(t)$

Eqns.(4) and (5) can be combined and stated as follows:

$$x(t) \leq (C(t) + B) \wedge ((x \otimes \beta)(t) + b) \quad \forall t \geq 0$$

Let $x_{\max}(t)$ be the maximum value of $x(t)$ which satisfies the above inequality. This inequality is of the form:

$$x \leq (C + B) \wedge \Pi(x) \tag{6}$$

where $x$ and $C$ are functions and $\Pi$ is an operator given by $\Pi(x) = (x \otimes \beta) + b$. It follows from [4] (see also [2] for further details), that the maximum solution which satisfies Eqn.(6) is given by $x_{\max}(t) = \overline{\Pi}(C(t) + B)$, where $\overline{\Pi}$ is the sub-additive closure of $\Pi$ and is defined as

$$\overline{\Pi}(x) = x \wedge \Pi(x) \wedge \Pi(\Pi(x)) \wedge \ldots$$

Since $\Pi(x) = (x \otimes \beta) + b$, it follows that:

$$\overline{\Pi}(x) = x \wedge (x \otimes \beta + b) \wedge (x \otimes \beta \otimes \beta + 2b) \wedge \ldots$$

or, $\overline{\Pi}(x) = x \wedge \inf_{n\geq 1}\{x \otimes \beta^{(n)} + nb\}$, where $\beta^{(n)}$ is the $n$-th self-convolution of $\beta$. Now, it is known that for any functions $f$, $g$ and $h$, $(f \wedge g) \otimes h = (f \otimes h) \wedge (g \otimes h)$ [4]. Using this result, it follows that: $\overline{\Pi}(x) = x \otimes \delta_0 \ \wedge \ x \otimes \inf_{n\geq 1}\{\beta^{(n)} + nb\}$, where $\delta_0$ is a function defined as $\delta_0(t) = +\infty$ for all $t > 0$, and $\delta_0(t) = 0$ for all $t \leq 0$. Hence, $\overline{\Pi}(x) = x \otimes \inf_{n\geq 0}\{\beta^{(n)} + nb\}$ since, for any function $f$, by convention $f^{(0)} = \delta_0$ (see [4]).

The sub-additive closure of any function $f$, denoted by $\bar{f}$, is defined as $\bar{f} = \inf_{n\geq 0}\{f^{(n)}\}$ (which is similar to the sub-additive closure of an *operator* as described above). Hence, it follows that $\overline{\Pi}(x) = x \otimes \overline{(\beta + b)}$ and therefore,

$$x_{\max}(t) = (C(t) + B) \otimes \overline{(\beta(t) + b)} \tag{7}$$

Similarly, to obtain a lower bound on $x(t)$, we recast Eqn.(5) as follows: $x(t) \geq (x(t) - b) \oslash \beta(t)$. By combining this with Eqn.(3), we obtain: $x(t) \geq (C \oslash \beta)(t) \ \vee \ (x(t) - b) \oslash \beta(t)$. This is of the form:

$$x \geq (C \oslash \beta) \vee \Gamma(x) \tag{8}$$

where $\Gamma$ is an operator given by $\Gamma(x) = (x - b) \oslash \beta$. Using a result [4] analogous to the existence of the maximum solution to Eqn.(6), it follows that Eqn.(8) has one minimum solution which is given by:

$$x_{\min}(t) = \underline{\Gamma}(C(t) \oslash \beta(t)) \tag{9}$$

where $\underline{\Gamma}$ is the super-additive closure of $\Gamma$ and is defined as

$$\underline{\Gamma}(x) = x \vee \Gamma(x) \vee \Gamma(\Gamma(x)) \vee \ldots$$

Unlike Eqn.(7), Eqn.(9) unfortunately does not give a closed-form solution to $x_{\min}(t)$ and must be iteratively computed for any given problem instance. Eqns.(7) and (9) therefore give upper and lower bounds on the function $x(t)$, and this is summarized in the following theorem, which is the main result of this paper.

**Theorem 1** *Any non-decreasing function $x(t)$ which satisfies the inequality: $x_{\min}(t) \leq x(t) \leq x_{\max}(t), \ \forall t \geq 0$, respects both, buffer overflow and the playout buffer underflow constraints, where $x_{\min}$ and $x_{\max}$ are computed using Eqns.(7) and (9).*

### D. The case of multiple processing elements

PEs in the path of a stream, other than those considered in the preceding subsections (i.e. those which do not feed their output into a playout buffer) process an input stream $x'(t)$ and the output $y'(t)$ is fed into another PE for further processing. The only constraint for any such PE is that the associated internal buffer should not overflow. The required output $y'(t)$ for such a PE is determined from the input $x(t)$ of the preceding PE. Following this *composition scheme*, we first fix an input $x(t)$ of the PE which feeds the playout buffer (where $x_{\min}(t) \leq x(t) \leq x_{\max}(t)$ from the previous subsection). This $x(t)$ is the required output $y'(t)$ of the immediate next PE, for which we compute bounds $x'_{\max}$ and $x'_{\min}$ (using similar techniques as described above) and choose some $x'(t)$ lying in between these bounds. This is the required output of the immediate next PE, and this process is followed until we compute $x'(t)$ (or bounds on it) for the input stream entering the first PE in the path of the stream. Any input stream conforming to this computed value is guaranteed to respect all the buffer constraints.

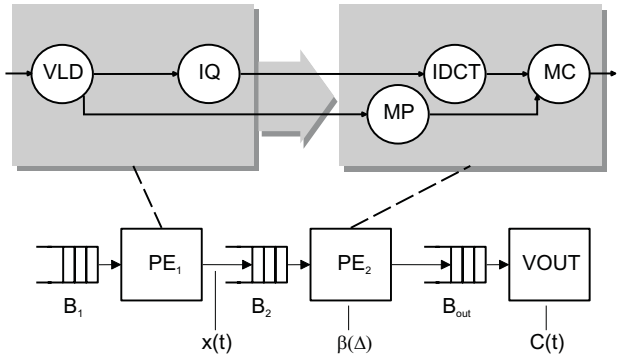


Fig. 2. Mapping the MPEG-2 decoder application onto a multiprocessor SoC.

## III. Rate Analysis for a MPEG-2 Decoder

In this section we apply the rate analysis methodology developed in the previous section to study the mapping of a MPEG-2 decoder [3] application onto a multiprocessor SoC architecture with fixed buffers. By comparing the results from our mathematical framework with those obtained from a system simulator, we show that our framework is able to provide useful bounds on the allowed rates of the input stream. For any input sequence $x(t)$ and the computed bounds $x_{\max}$ and $x_{\min}$, the results obtained by our framework were in conformance with the simulation results in terms of predicting buffer overflow, underflow, and all cases where the buffer constraints are satisfied.

### A. The MPEG-2 decoder application

Our target architecture consists of a number of PEs interconnected by an on-chip communication network. This network can be seen as a system of point-to-point buffered FIFO channels of limited capacity. The PEs exchange packetized data by writing/reading to/from these channels.

The part of the architecture onto which the MPEG-2 decoder application is mapped, along with the mapping of the application's task graph on it is shown in Figure 2. In this figure, $PE_1$ and $PE_2$ are programmable processors and $VOUT$ denotes a video output port.

The task graph of the MPEG-2 decoder includes several tasks such as variable length decoding (VLD), inverse quantization (IQ), inverse discrete cosine transform (IDCT), formation of motion predictors (MP) and motion compensation (MC). Based on profiling information, we partitioned this set of tasks into two subsets, with one being executed on $PE_1$, and the other on $PE_2$.

A compressed video bit stream arrives into a buffer $B_1$ (as shown in Figure 2) and is processed by the VLD and the IQ tasks running on $PE_1$. Decompressed *macroblocks* are written into the buffer $B_2$, which is read by $PE_2$ for further processing. Note that the data exchange between $PE_1$ and $PE_2$ can be seen as a single stream of packets with each packet encapsulating IDCT coefficients and motion vectors for one macroblock. This information is processed by the IDCT, MP and MC tasks mapped onto $PE_2$. Finally, the decoded video samples are written (one macroblock at a time) into the playout buffer $B_{out}$.

$B_{out}$ is read by an *output process* located in the video port, which reads one macroblock at a time, at a constant rate that is determined by the frame rate and the resolution of the decoded video stream. The constraints associated with $B_{out}$ are that it should never be empty when the output process attempts to read from it, and it should also not overflow when $PE_2$ writes into it. We would not want adopt the option of stalling $PE_2$ when $B_{out}$ is filled, in order to use simple static scheduling algorithms on $PE_2$ when multiple streams are processed by it. Additionally, we also require the buffers $B_1$ and $B_2$ not to overflow.

Satisfying all the above conditions simultaneously is difficult, because tasks executing on $PE_1$ produce a bursty time-varying traffic on its output. This is mainly due to the fact that the execution time of the VLD task exhibits high variability, which depends on the structure of the compressed stream and the properties of the encoded video information (see also [20]).

Using the proposed rate analysis technique, we can compute upper and lower bounds on the macroblock output rate that is to be satisfied by the processor $PE_1$. Any macroblock output stream from $PE_1$ which conforms to these bounds is guaranteed not to overflow the buffers $B_2$ and $B_{out}$ and also not underflow the buffer $B_{out}$. Depending on such a chosen output rate of $PE_1$, upper and lower bounds on the input rate to the buffer $B_1$ can also be computed, as discussed in the previous section.

### B. Analytical results

Due to space restrictions, we will only concentrate on buffer overflow and underflow constraints associated with $B_2$ and $B_{out}$ and compute bounds that the macroblock output stream coming out of $PE_1$ is required to satisfy. Following the notation of Section II (see also Figure 1), the output process located in the video port is a *real-time client* characterized by a cumulative rate function $C(t)$. $B_{out}$ is the playout buffer of size $B$. The processing capacity of $PE_2$ is characterized by a guaranteed service of at least $\beta(t)$. The buffer $B_2$ corresponds to the internal buffer of $PE_2$ with size $b$. Let $x(t)$ be the cumulative rate function of any macroblock stream on the output of $PE_1$. The curves denoted by $x_{max}(t)$ and $x_{min}(t)$, which are computed analytically using the framework developed in Section II, are the bounds which the macroblock stream $x(t)$ on $PE_1$'s output has to conform in order to guarantee that buffers $B_2$ and $B_{out}$ do not overflow and $B_{out}$ does not underflow.

$C(t)$ can be derived from the parameters of the video stream to be decoded, and is given as follows:

$$C(t) = \begin{cases} 0 & \text{if } t \leq t_0 \\ NFt & \text{if } t > t_0 \end{cases}$$

where $N$ is the number of macroblocks per video frame, $F$ is the video frame rate and $t_0$ is the time, starting from which the real-time client starts reading data out of the playout buffer. $t_0$ can be referred to as the *playback delay* or *buffering time*. All video streams used in our experiments have $F = 25$ *frames per second* and $N = 1620$. The playback delay $t_0$, in general, can be chosen arbitrarily. We have set $t_0$ to be equal to the time required by the real-time client to read half a frame from the playout buffer.

A *system configuration* is defined by a set of parameters for $\beta(t)$, $B$, and $b$. In our experiments we varied the size of the playout buffer $B$. Assuming that the processor $PE_2$ is unloaded and hence its entire capacity is available for processing

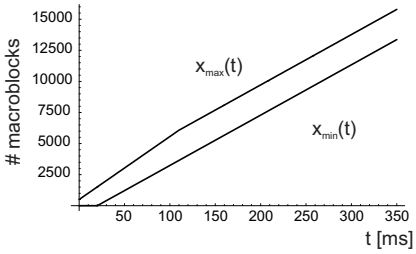


Fig. 3. The analytical bounds $x_{min}(t)$ and $x_{max}(t)$ computed for a system configuration with $B = 2430$ macroblocks.

the IDCT, MP and MC tasks mapped onto it, the service curve $\beta(t)$ was modeled by a straight line. The slope of this line was set to be equal to the long-term average macroblock production rate on the output of $PE_1$ (which was measured using the system simulator described in the next subsection). This was based on the assumption that in the long term, $PE_2$ has sufficient capacity to process all the incoming macroblocks. $B_2$ was always set to a fixed size of $b = 500$ macroblocks. Figure 3 shows the resulting bounds $x_{min}(t)$ and $x_{max}(t)$ computed by implementing Eqns. (7) and (9) of Section II in Mathematica (from Wolfram Research), with the system configuration and $\beta(t)$ described above and the playout buffer size $B$ set to 2430 macroblocks.

### C. Simulation Setup

We performed simulations of the MPEG-2 decoder application (shown in Figure 2) using a transaction level model of the system architecture (see [12]). The system model was written in SystemC [17], and the models of the programmable PEs were based on the Sim-Profile configuration of the SimpleScalar [1] instruction set simulator. Both, $PE_1$ and $PE_2$, had a RISC-core (similar to the MIPS3000 processor) augmented with MPEG-2 specific hardware accelerators. $PE_1$ was enhanced with bit-stream access operations, while $PE_2$ had special support for application kernels such as IDCT, Add_Clip and Block_Average and could prefetch memory in a special video-block mode. Floating point operations were not used on either of the PEs. The implementation of the MPEG-2 decoder was based on the source code available from [15].

We simulated the decoding of several MPEG–2 video clips. All video clips had parameters as described in the previous subsection. The clips were encoded using a constant bit rate of 9.78 Mb/s and a resolution of 720x576 pixels (typically used in DVD applications). A selection of the simulated scenarios (with each scenario being a combination of a video clip and the playout buffer size), representing corner cases for our experiments, is summarized in Table I. Video sequence A corresponds to a video clip with global motion, whereas video sequence B corresponds to a video clip with moving objects and still background and video sequence C represents a still picture.

For all the simulation scenarios listed in Table I, using our simulation setup we *measured* $x(t)$ at the output of $PE_1$ and the maximum and minimum *fill levels* of $B_2$ and the playout buffer $B_{out}$.

### D. Comparing the analytical bounds with simulation

In this subsection we evaluate the usefulness of the analytical bounds on the input rate $x(t)$, by comparing the predictions

TABLE I
SIMULATION SCENARIOS

| Scenario | Buffer Size B # macroblocks | Video Clip |
|---|---|---|
| 1 | 1620 | Sequence A |
| 2 | 2430 | Sequence A |
| 3 | 2430 | Sequence B |
| 4 | 2430 | Sequence C |
| 5 | 3240 | Sequence C |

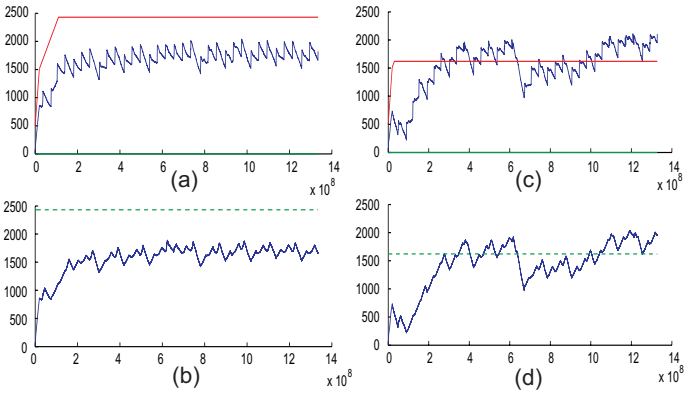


Fig. 4. (a) The *difference plot* for a macroblock stream $x(t)$ which is compliant with the computed upper and lower bounds; (b) Corresponding playout buffer fill levels; (c) The *difference plot* for a non-compliant stream $x(t)$; (d) Corresponding playout buffer fill levels indicating buffer overflow. The horizontal-axis shows time in ns and the vertical-axis shows the number of processed macroblocks in the playout buffer.

on buffer overflow/underflow/conformance deduced from the analytical framework, with the results obtained by simulation.

For the ease of interpretation of the simulation results, we always show a *difference plot*. Such a plot does not show the absolute values of $x(t)$ (obtained from the simulation) versus $x_{min}(t)$ and $x_{max}(t)$ (which are computed following Theorem 1 of Section II). Instead, it shows the curves corresponding to the differences $x_{max}(t) - x_{min}(t)$ and $x(t) - x_{min}(t)$. From such a plot it is possible to detect when an input stream $x(t)$ (where $x(t)$ is measured from the input stream resulting from simulating the decoding algorithm on a video clip) violates the computed bounds. A violation occurs whenever the curve representing $x(t) - x_{min}(t)$ crosses the curve $x_{max}(t) - x_{min}(t)$, or goes below 0.

Figure 4(a) shows an example where $x(t)$ resulting from a video clip is compliant with the bounds $x_{min}(t)$ and $x_{max}(t)$. In Figure 4(b) the corresponding playout buffer fill level (as measured from simulation) is shown, which confirms that no buffer overflow or underflow occurs. Figure 4(c) depicts an example of a sequence $x(t)$ (obtained from the simulation Scenario 1), which violates the upper bound. The corresponding buffer fill level plot in Figure 4(d) shows that the playout buffer overflows.

In Figure 5 we show the difference plots corresponding to the Scenarios 2–5 which are outlined in Table I. All the figures in this subsection show an excerpt of 1.36 seconds (corresponding to 34 frames) of video sequences from some simulation scenario (1 to 5). The left bar plot in Figure 6 shows normalized values of largest buffer fill levels observed at the play-

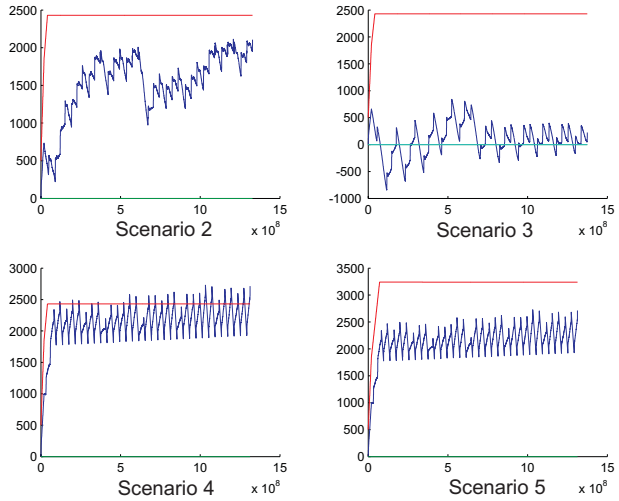


Fig. 5. Difference plots ($x_{\max}(t) - x_{\min}(t)$ and $x(t) - x_{\min}(t)$) for the simulation Scenarios 2–5, with the corresponding normalized buffer fill and emptiness levels of the playout buffer shown in Figure 6. The horizontal-axis shows time in ns and the vertical-axis shows the number of processed macroblocks in the playout buffer.

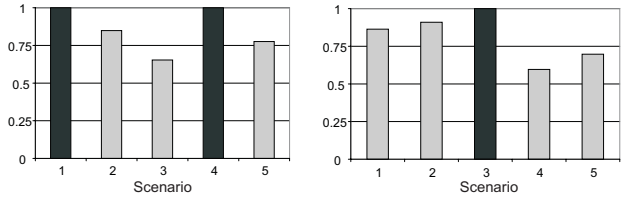


Fig. 6. Normalized buffer fill levels (on the left) and emptiness levels (on the right) for the simulation Scenarios 1 to 5. The bars coloured black indicate buffer overflows and underflows in the respective bar graphs.

out buffer. On the right, the normalized "emptiness" values $E$ (or smallest buffer fill levels) are shown for the simulation runs. We define normalized emptiness as $E = \sup_{\forall t > t_0}\{\frac{B - fill(t)}{B}\}$, where $fill(t)$ denotes the actual buffer fill level at time $t$ and $t_0$ is the playback delay.

For Scenarios 1 and 4, we can observe that $x(t)$ measured from simulations is greater than the computed $x_{\max}(t)$ for some values of $t$. The measured buffer fill levels for both these scenarios show overflow, which is in agreement with the analytical framework. Similarly for scenario 3, we see that $x(t)$ is less than $x_{\min}(t)$ for some values of $t$, indicating a possibility of the playout buffer underflow, which is confirmed in Figure 6. For the remaining two scenarios (i.e. 2 and 5), $x(t)$ was measured to be within the computed bounds and no buffer overflow or underflow was observed.

These experiments therefore indicate that the proposed analytical framework provides meaningful bounds on the rate of the input stream, which are in conformance with detailed simulation results.

## IV. Concluding Remarks

We presented a novel framework for *rate analysis* for streaming applications, and showed its feasibility through a case study of mapping a MPEG–2 decoder application onto a multiprocessor SoC architecture. This framework can be used to determine the admissible input stream rates with which a system running a stream processing application can be loaded. The main application of such an analysis is in system-level performance evaluation of hardware-software architectures for stream processing, and can be used evaluate different possible mappings of an application onto a fixed architecture. In contrast to simulation based approaches, our framework can be used to meaningfully evaluate a large number of such mapping within a very short time and can therefore be used for automated design space exploration techniques. We plan to integrate this framework into appropriate tools, and also explore its use in on-chip buffer sizing and scheduling for streaming applications.

**Acknowledgements:** The first two authors are partly supported by the Swiss Innovation Promotion Agency (KTI/CTI) through the projects KTI 5845.1 and KTI 5500.2. The work of the third author is partly supported by the NUS ARF grants R-252-000-169-101/112.

## References

[1] T. Austin, E. Larson, and D. Ernst. SimpleScalar: An infrastructure for computer system modeling. *IEEE Computer*, 35(2):59–67, 2002.

[2] F.L. Baccelli, G. Cohen, G.J. Olsder, and J.-P. Quadrat. *Synchronization and Linearity: An Algebra for Discrete Event Systems*. John Wiley & Sons, 1992.

[3] V. Bhaskaran and K. Konstantinides. *Image and Video Compression Standards: Algorithms and Architectures*. Kluwer Academic Publishers, 1997.

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

[5] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *6th Design, Automation and Test in Europe (DATE)*, 2003.

[6] S. Chakraborty, S. Künzli, L. Thiele, A. Herkersdorf, and P. Sagmeister. Performance evaluation of network processor architectures: Combining simulation with analytical estimation. *Computer Networks*, 41(5), 2003.

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

[8] R. Cruz. Quality of service guarantees in virtual circuit switched networks. *IEEE Journal of Selected Areas in Communication*, 13(6), 1995.

[9] W. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. In *38th Design Automation Conference (DAC)*, 2001.

[10] M.I. Gordon et al. A stream compiler for communication-exposed architectures. In *10th Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 291–303, 2002.

[11] M. Gries, C. Kulkarni, C. Sauer, and K. Keutzer. Comparing analytical modeling with simulation for network processors: A case study. In *Proc. of the Designer's Forum at DATE*, 2003.

[12] A. Haverinen, M. Leclercq, N. Weyrich, and D. Wingard. SystemC based SoC Communication Modeling for the OCP Protocol. `http://www.ocpip.org`, October 2002.

[13] K. Lahiri, A. Raghunathan, and S. Dey. Evaluation of the traffic-performance characteristics of system-on-chip communication architecutres. In *Intl. Conf. on VLSI Design*, pages 21–35, 2001.

[14] K. Lahiri, A. Raghunathan, and S. Dey. System level performance analysis for designing on-chip communication architectures. *IEEE Trans. on Computer Aided-Design of Integrated Circuits and Systems*, 20(6):768–783, 2001.

[15] MPEG Software Simulation Group. `http://www.mpeg.org`.

[16] K. Richter, M. Jersak, and R. Ernst. A formal approach to MpSoC performance verification. *IEEE Computer*, 36(4), 2003.

[17] Open SystemC Initiative. `http://www.systemc.org`.

[18] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli. A framework for evaluating design tradeoffs in packet processing architectures. In *39th Design Automation Conference (DAC)*, 2002.

[19] W. Thies, M. Karczmarek, and S. Amarasinghe. StreamIt: A language for streaming applications. In *11th Conference on Compiler Construction (CC)*, LNCS 2304, pages 179–196, 2002.

[20] G. Varatkar and R. Marculescu. Traffic analysis for on-chip networks design of multimedia applications. In *39th Design Automation Conference (DAC)*, 2002.