Processor Frequency Selection for SoC Platforms for Multimedia Applications

Yanhong Liu[1] Alexander Maxiaguine[2] Samarjit Chakraborty[1] Wei Tsang Ooi[1]
[1]Department of Computer Science, National University of Singapore
[2]Computer Engineering and Networks Laboratory, ETH Zürich
E-mail: {liuyanho, samarjit, ooiwt}@comp.nus.edu.sg, maxiagui@tik.ee.ethz.ch

Abstract

Of late, there has been a considerable interest in generic and configurable System-on-Chip platforms specifically targeted towards implementing multimedia applications. A number of such platforms offer the possibility of including processor soft cores which are highly customizable. For voltage/frequency scaled processors, such customization includes the selection of appropriate voltage/frequency operating points which are tuned to the application set to be mapped onto the platform. In this context, we present an analytical framework that can guide a system designer in identifying the frequency ranges that should be supported by the different processors of a platform architecture. This framework can also be used to identify how such frequency ranges depend on the different parameters of the architecture (such as on-chip buffer sizes), and the performance impacts associated with selecting a particular frequency range. In the case of multimedia streaming applications, identifying such performance impacts and tradeoffs involved in customizing a platform architecture is especially difficult due to the bursty nature of on-chip traffic arising out of multimedia processing and the high variability in their execution requirements. The framework presented here is designed to precisely capture such characteristics and can be used in the design-space exploration of energy-aware platform architectures for multimedia processing.

1 Introduction

Today, most portable devices such as PDAs and mobile phones offer support for streaming multimedia applications. Because of flexibility, time-to-market advantages and low design costs, very often such devices are now designed using generic, configurable System-on-Chip (SoC) platforms. Such platforms are usually customized for a particular application domain, but still support sufficient flexibility to allow them to be configured for specific products belonging to that domain. Examples of such platforms are the Eclipse architecture template [10] and the Viper SoC architecture [5] from Philips, both of which target advanced set-top box and DTV markets.

Because of the high computational demands, real-time constraints and low power consumption requirements of portable multimedia devices, *designing* SoC platforms for such devices require a disciplined design methodology. Although, recently there has been some work on *platform management* techniques [13], the issue of *platform design* has not been sufficiently addressed so far. The only significant effort in this direction, that we are aware of, is from Philips in designing the Eclipse architecture template for media processing SoCs [11, 12]. In this paper, we follow this line of work and propose a framework for designing energy-aware SoC platforms specifically targeted towards media processing in portable devices. In the application domain targeted by Eclipse (digital televisions and set-top boxes), power consumption is not a significant issue.

Problem Statement: The problems we are interested in addressing are of the following form. Suppose that we are given a multiprocessor SoC platform architecture "template" and a number of multimedia applications, all of which are required to be supported by this platform. Our job is to derive a (concrete) platform architecture from this template, by choosing appropriate processors, sizes of on-chip buffers and possibly other parameters such as bus widths and cache configurations. The processors to be chosen for this platform support software-controlled voltage and frequency scaling to allow different degrees of power consumption at run time. Therefore, we are also required to choose the frequency/voltage ranges that each processor should support. In this paper we specifically focus on this last issue and identify how this range depends on the other parameters of the platform architecture, such as on-chip buffer sizes.

The results presented in this paper also provide insights into questions such as: if a processor supports only a fixed number of *operating points*, where each such point is characterized by a voltage and a frequency value, then how many such operating points should a processor ideally support and how should these values be chosen? A processor which allows the voltage and frequency values to be changed continuously would typically be more expensive than one which allows these values to be changed in discrete steps and supports only a fixed number of these val-

Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS 2004)
1052-8725/04 $20.00 © 2004 **IEEE**

ues or operating points. Today, processors of both these types are available—Intel's XScale processor is of the former type and Transmeta's Crusoe processor is of the latter type. Therefore, it is pertinent to ask questions like what kind of performance impacts would choosing a processor of the latter type have, over a more expensive processor which supports a continuous range of frequency values? Further, a platform designer would also be interested in identifying how the frequency range, that needs to be supported by a processor, varies with the available on-chip buffer size. Since on-chip buffers are available only at a premium because of their high area requirements [14], such information would help in choosing an appropriate tradeoff.

Answers to these questions are becoming increasingly relevant because a number of embedded processor cores today offer a high degree of customization potential, such as instruction set tailoring and register file sizing. For processors which support dynamic frequency/voltage scaling, choosing efficient operating points is therefore becoming a part of this customization procedure. In the context of multimedia applications, this is especially critical because of the complex and bursty nature of on-chip traffic and the high variability in the execution times of multimedia processing tasks—both of these resulting in a highly variable demand on the computational resources available on the chip [14]. Hence, being able to control the processor frequency accurately to counter this variability is important.

Our Results and Relation to Previous Work: The main contribution of this paper is framework which can guide a system designer in identifying the operating frequency range that different processors on a SoC platform architecture should support in order to run a given multimedia application or a class of applications. Identifying such a range accurately is not straightforward because of the reasons mentioned above, i.e. the complex nature of on-chip traffic arising out of multimedia processing and the variability in the execution times of tasks. Moreover, since different applications and input classes might have very different computational demands, choosing an appropriate processor frequency range involves several tradeoffs between processor cost, flexibility and on-chip buffer requirements. Our framework can help a system designer in identifying these tradeoffs.

Although there has been a significant amount of work in developing voltage and frequency scheduling algorithms in the context of multimedia applications, the problem of processor design and processor frequency selection from an energy-aware perspective has received considerably little attention so far. Our work is partly motivated by a recent paper [2], which proposes a linear programming based technique to optimally select operating voltage/frequency points in an embedded processor core. The task to be executed on such a processor is specified as a task graph whose vertices are annotated with execution requirements and deadlines. However, in contrast to this work, the framework proposed here is targeted specifically towards streaming multimedia applications and explicitly models the burstiness in multimedia streams and the variability in the execution requirements of multimedia processing tasks. Other representative work in this direction is [6], which addresses the selection of the processor core and instruction and data cache configuration in the design of variable voltage processors.

The framework presented in this paper is based on the theory of *Network Calculus* [1] which was developed and is still largely used in the context of analyzing communication networks. Very recently, it was extended to analyze SoC architectures in the context of network processors [4]. This work was further extended in [3, 9] to the domain of general SoC platform architectures. Our work in this paper follows this line of development, but extends the underlying theory in two ways. On an abstract level, the analysis schemes presented in the previous papers relied on concrete input instances and could not provide any theoretical guarantees on the performance of an architecture for a *class* of inputs. In contrast to this, the framework presented here can be used to analyze a *class* of input streams, for which a more elaborate theory is necessary—this is explained in detail in Sections 3.1 - 3.3. Secondly, none of the previous results provided means for computing the range of processor frequencies from an input specification. This extension is presented in Section 4.

The rest of the paper is organized as follows. The next section describes a system model of a platform architecture and formally states our problem. Given a specification of the application to be implemented on this architecture and the class of input streams to be processed, in Section 3 we compute bounds on the *service* that needs to be provided by each processor of this architecture. In Section 4 we show how such service bounds can be used to derive the operating frequency range of each processor. Finally, in Section 5 we present a case study involving an MPEG-2 decoder application to illustrate an application of the proposed framework, and also validate the results obtained using detailed simulations.

2 Problem Formulation

In this paper, we consider the following system-level view of multimedia stream processing on a SoC platform. A platform architecture, such as the one shown in Figure 1, consists of multiple processing elements (PEs) onto which the different parts of an application are partitioned and 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. A model of the architecture in Figure 1 is shown in Figure 2. Each PE has an internal buffer, which is a FIFO channel of fixed capacity, and is used to store the incoming stream to

Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS 2004)
1052-8725/04 $20.00 © 2004 **IEEE**

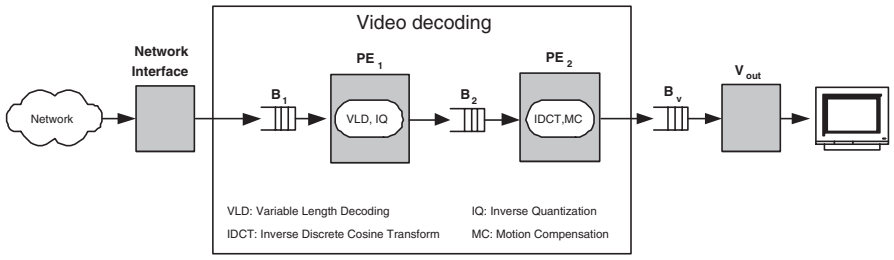

Figure 1. A multiprocessor SoC platform onto which an MPEG-2 decoder application is partitioned and mapped.

be processed. Finally, the fully processed stream is written into a *playout buffer* which is read by some *real-time client* (RTC) such as an audio or a video output device.

For the sake of generality, we consider any multimedia stream that is seen by a PE belonging to the platform, to be made up of a potentially infinite 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. For example, in the architecture shown in Figure 1, stream objects entering PE_1 are single bits. But stream objects entering PE_2 are coded macroblocks and the stream objects being written into the playout buffer B_v are decoded macroblocks.

In Figure 2, let $x_i(t)$ denote the number of stream objects that arrive at a PE PE_i during the time interval $[0, t]$. Let $y_i(t)$ (equal to $x_{i+1}(t)$) denote the number of processed stream objects at the output of PE_i (or the input of PE_{i+1}) during the time interval $[0, t]$. The real-time client RTC consumes stream objects from the playout buffer at a rate $C(t)$, which again denotes the number of stream objects consumed within the time interval $[0, t]$. The stream entering the processing element PE_i receives a *service* β_i, which is specified by a tuple (β_i^l, β_i^u). Within *any* time interval of length Δ, it is guaranteed that PE_i will process *at least* $\beta_i^l(\Delta)$ number of stream objects and it will be able to process *at most* $\beta_i^u(\Delta)$ number of stream objects. The functions β_i^l and β_i^u therefore represent lower and upper bounds on the service provided by PE_i and is determined by the time required to process each stream object, the scheduling policy implemented on this PE (in case multiple streams are being processed by it), and also by the voltage/frequency scheduling policy implemented on it. Lastly, each PE PE_i is also associated with a tuple $\gamma_i = (\gamma_i^l, \gamma_i^u)$, where $\gamma_i^l(k)$ and $\gamma_i^u(k)$ denote the minimum and the maximum number of *processor cycles* respectively, that may be required to process *any* k consecutive stream objects belonging to the input stream. γ_i is therefore used to capture the variability in the execution requirements of the different stream objects.

Now recall from Section 1 that for each PE belonging to the platform, we would like to determine the operating frequency range that should be supported by it. If the processor supports only a fixed number of discrete frequency levels, then we would like to determine how should these frequencies be chosen and what kind of performance impacts will this decision have. Note that the platform should be designed to support a *class* (or several classes) of multimedia streams. For example, a portable multimedia device might have a wireless interface through which MPEG-2 coded video streams of two different classes come in—high-quality video clips with 8 Mbps input bit rate and low-quality clips with 4 Mbps input bit rate. The computational demands associated with these two input classes might vary widely, which translates to different operating frequency requirements for any PE on the platform. The input to a PE, when specified using the function $x_i(t)$, however, represents a concrete instance of a stream rather than a *class* of streams. Therefore, to specify the arrival pattern of a class or family of streams, we use an abstraction called *arrival curve* which is similar to the concept of *service* β_i described above. The arrival curve α_{x_i} representing the class of streams that might arrive at the input of PE_i is also specified by a tuple $(\alpha_{x_i}^l(\Delta), \alpha_{x_i}^u(\Delta))$, where the first and the second terms represent the minimum and the maximum number of stream objects that might arrive within any time interval of length Δ. In other words, $\alpha_{x_i}^l(\Delta) \leq x_i(t+\Delta) - x(t) \leq \alpha_{x_i}^u(\Delta),\ \forall t, \Delta \geq 0$. Therefore, any concrete arrival pattern $x_i(t)$ is lower and upper bounded by the functions $\alpha_{x_i}^l$ and $\alpha_{x_i}^u$ respectively. Similarly, we use $\alpha_{y_i}^l$ and $\alpha_{y_i}^u$ to denote lower and upper bounds on the arrival pattern of the processed stream at the output of PE_i.

Now, let us consider the last PE in the path of a stream, i.e. the PE whose output is written into the playout buffer (see Figure 2). Henceforth, for simplifying the notation, we drop the subscript i representing the PE identifier. Therefore, as described above, any input instance to this PE is specified by the function $x(t)$ and the class of all input instances is bounded by the arrival curve α_x. Any output arrival pattern from this PE is represented by the function $y(t)$ and the sizes of the internal and the playout buffers are b and B respectively. The consumption pattern of stream objects from the playout buffer is specified by the function $C(t)$ described above. Now, given $\alpha_x(\Delta)$, $\gamma(k)$, $C(t)$ and the buffer sizes b and B, the problem is to compute the set of all possible processor frequencies at which this PE might be run, such that the following constraints are satisfied: (i) the

Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS 2004)
1052-8725/04 $20.00 © 2004 **IEEE**

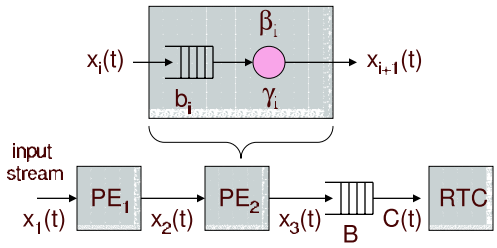

Figure 2. System-level view of multimedia processing on a multiprocessor SoC platform.

playout buffer never overflows, (ii) it never underflows, and (iii) the internal buffer never overflows. The constraint on the playout buffer underflow is to ascertain that stream objects can be read out by the output audio/video devices (the RTC in Figure 2) at the rate specified by $C(t)$, and hence the output quality is guaranteed. The constraints on buffer overflow are motivated by the fact that typically on-chip PEs use static voltage and task scheduling policies. This is because using blocking write/read mechanisms efficiently to prevent buffer overflows/underflows either require a multithreaded processor architecture or substantial run-time operating system support for context switching.

Our solution to the above problem consists of two parts. In Section 3 we compute lower and upper bounds on the service (β^l and β^u) that needs to be provided by the PE in order to satisfy the above mentioned buffer constraints. In Section 4, we then show how to compute the frequency range that needs to be supported by the PE in order to realize these service bounds. The extension of these results to any other PE in the path of a stream (i.e. one whose output is not written into the playout buffer, but instead into another PE) is fairly simple, and is also explained in Section 3.4.

Throughout this paper we assume the following processor model: a PE can either support a continuous range of clock frequencies, or a fixed number of discrete frequencies, where this number can also be equal to one, i.e. the PE runs at a fixed frequency and does not support frequency scaling. Any clock frequency is associated with a minimum operating voltage that needs to be supplied to run the processor at this frequency. We assume that this is the voltage at which the processor is run for any frequency, i.e. voltage and frequency are tightly coupled and determining the frequency results in the voltage also being determined. Hence, we will only be concerned with determining the frequency range or the discrete frequency values for any given PE.

3 Bounds on Service Requirements

Given $\alpha_x(\Delta)$, $C(t)$ and the buffer sizes b and B for the last PE in the path of a stream, in this section we compute the lower and the upper bounds $\beta^l(\Delta)$ and $\beta^u(\Delta)$ on the service that needs to be provided to this stream to satisfy the buffer overflow and underflow constraints described in Section 2. Within any interval of length Δ, if the service provided is less than our computed $\beta^l(\Delta)$, then either the internal buffer might overflow or the playout buffer might underflow. Similarly, if the service provided is greater than the computed $\beta^u(\Delta)$, then the playout buffer might overflow. Following the notation introduced in Section 2, we use $x(t)$ to denote any arrival pattern of stream objects at the input of the PE and $y(t)$ to denote the arrival pattern at the output of the PE. Recall that the functions x, y and C always denote *cumulative values* over the time interval $[0, t]$, whereas the functions α_x and β take *time interval lengths* as the input parameter.

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: $(f \oslash g)(t) = \sup_{u:u\geq 0}\{f(t+u) - g(u)\}$. The *max-plus convolution* of f and g is given by: $(f \overline{\otimes} g)(t) = \sup_{s:0\leq s\leq t}\{f(t-s)+g(s)\}$. 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.

Let us assume that at time $t = 0$, the buffer fill-levels of the playout buffer and the internal buffer of the last PE in the path of the stream are B_0 and b_0 respectively (see [8] for an explanation). Then the constraint on the playout buffer underflow can be stated as (see Figure 2):

$$y(t) \geq C(t) - B_0, \quad \forall t \geq 0 \tag{1}$$

Similarly, the constraint on the playout buffer overflow can be stated as: $y(t) \leq C(t) + B - B_0, \quad \forall t \geq 0$ (2)

Finally, the constraint that the internal buffer in the PE should not overflow, is given by:

$$y(t) \geq x(t) - (b - b_0), \quad \forall t \geq 0 \tag{3}$$

The constraints (1) and (3) can be combined and stated as: $y(t) \geq (C(t) - B_0) \vee (x(t) - (b - b_0)), \forall t \geq 0$. Now, if $\beta^l(\Delta)$ is the minimum number of stream objects that is guaranteed to be processed by this PE within any time interval of length Δ, then it can be shown that $y(t) \geq (x \otimes \beta^l)(t)$, $\forall t \geq 0$ [1]. Hence, $(x \otimes \beta^l)(t)$ is the minimum value of $y(t)$ for any t, and therefore the above constraint on $y(t)$ can be reformulated as: $\forall t \geq 0$,

$$(x \otimes \beta^l)(t) \geq (C(t) - B_0) \vee (x(t) - (b - b_0)) \tag{4}$$

It can be shown [1] that for any functions f, g and h, $g \otimes h \geq f$ if and only if $h \geq f \oslash g$. Using this result, inequality (4) can be reformulated as: $\forall t \geq 0$,

$$\beta^l(t) \geq ((C(t) - B_0) \vee (x(t) - (b - b_0))) \oslash x(t) \tag{5}$$

Inequality (5) therefore gives a lower bound on the service that needs to be provided by the PE in order to satisfy the playout buffer underflow and the internal buffer overflow constraints. If $\beta^u(\Delta)$ is the maximum number of stream objects that can be processed by the PE within any time interval of length Δ, then it can be shown that $y(t) \leq (x \otimes$

Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS 2004)
1052-8725/04 $20.00 © 2004 **IEEE**

$\beta^u)(t), \forall t \geq 0$ [1]. Hence, following the same reasoning as above, $(x \otimes \beta^u)(t)$ is the maximum value of $y(t)$ for any t, and by using this, the constraint (2) can be reformulated as:

$$(x \otimes \beta^u)(t) \leq C(t) + B - B_0, \quad \forall t \geq 0 \tag{6}$$

Given the functions $x(t)$, $C(t)$ and the values of B and B_0, we would like to determine maximum value of $\beta^u(t)$ for any t (or the largest possible function β^u), for which inequality (6) is satisfied. Such a β^u would give an upper bound on the service that can be provided by the PE, which will satisfy the playout buffer overflow constraint.

It may be noted here that in some cases $\beta^u(t)$ can be infinitely large. For example, consider the case where the arrival pattern of stream objects (given by $x(t)$) is exactly the same as the consumption pattern by the real-time client from the playout buffer, i.e. $x(t) = C(t)$ for all $t \geq 0$. In this case $\beta^u(t)$ can be infinitely large, since no matter how much service is provided by the PE, the playout buffer can never overflow. In fact, $\beta^u(t)$ can be infinitely large if $x(t) \leq C(t)+B-B_0$ for all $t \geq 0$ (this also follows directly from the definition of the min-plus convolution operator).

3.1 Service Bounds for a Class of Streams

The above bounds on β^l and β^u are based on a specific instance of the arrival pattern of a stream, i.e. $x(t)$. Hence, these bounds can only guarantee the buffer overflow and underflow constraints for this specific arrival pattern. However, we would like to derive the service bounds for a class of arrival patterns—i.e. all arrival patterns which are bounded by the arrival curve α_x.

Computing the Bound on β^l: For a concrete arrival pattern of stream objects given by $x(t)$, the bound on β^l as given by inequality (5) can be shown to be equivalent to: $\beta^l(t) \geq \max\{(C \oslash x)(t) - B_0,\ \alpha^u_x(t) - (b - b_0)\}, \forall t \geq 0$. Since $x(t) \geq \alpha^l_x(t)$ for all $t \geq 0$, for any function f, $(f \oslash x)(t) \leq (f \oslash \alpha^l_x)(t)$ for all $t \geq 0$. Hence, the above constraint on $\beta^l(t)$ may be reformulated as:

$$\beta^l(t) \geq ((C \oslash \alpha^l_x)(t) - B_0) \vee (\alpha^u_x(t) - (b - b_0)),\ \forall t \geq 0$$

Further, let us assume that the consumption pattern of stream objects from the playout buffer, as specified by the function $C(t)$ is lower and upper bounded by the arrival curve α_C, i.e. $\alpha^l_C(\Delta) \leq C(t+\Delta) - C(t) \leq \alpha^u_C(\Delta), \quad \forall t, \Delta \geq 0$. Then the above constraint on β^l can be finally stated as:

$$\begin{aligned} \beta^l(t) \quad \geq \quad & ((\alpha^u_C \oslash \alpha^l_x)(t) - B_0) \\ & \vee (\alpha^u_x(t) - (b - b_0)),\ \forall t \geq 0 \end{aligned} \tag{7}$$

Inequality (7) therefore provides a lower bound on the minimum service that needs to be provided by the PE, in order to satisfy the playout buffer underflow and the internal buffer overflow constraints, where all arrival patterns at the PE are bounded by α_x and all consumption patterns from the playout buffer are bounded by α_C.

Computing the Bound on β^u: The upper bound on the maximum service that can be provided by the PE is given by the largest possible function β^u which satisfies inequality (6). We know that any instance of an arrival pattern $x(t)$ at the input of the PE is upper bounded by α^u_x, and the lower bound on the consumption pattern of stream objects from the playout buffer is given by α^l_C. Hence, the constraint on β^u can be reformulated as: $(\alpha^u_x \otimes \beta^u)(t) \leq \alpha^l_C(t)+B-B_0,\ \forall t \geq 0$. Again, given the functions α^u_x, α^l_C and B, B_0, we would like to determine the largest function β^u which satisfies the above inequality. Towards this, note that if $\alpha^u_x(t) \leq \alpha^l_C(t) + B - B_0,\ \forall t \geq 0$, then $\beta^u(t)$ can be infinitely large for all $t > 0$. This corresponds to the case where the rate at which stream objects arrive at the PE is not high enough to overflow the playout buffer, irrespective of how fast they are processed.

Now, let us consider the case where, $\alpha^u_x(t_i) > \alpha^l_C(t_i) + B - B_0, \quad$ for $i = 1, \ldots, n$, and for all other values of t, $\alpha^u_x(t) \leq \alpha^l_C(t) + B - B_0$. Assuming that $t_i < t_{i+1}$ for $i = 1, \ldots, n-1$, the largest function β^u is then given as follows:

$$\beta^u(t) \leq \begin{cases} 0 & \text{if } t = 0, \\ \alpha^l_C(t_1) + B - B_0 & \forall t \leq t_1 \\ \alpha^l_C(t_{i+1}) + B - B_0 & \forall t_i < t \leq t_{i+1}, \\ & i = 1, \ldots, n-1 \\ \infty & \forall t > t_n \end{cases} \tag{8}$$

A proof of the above may be found in [8]. The above upper bound on β^u therefore guarantees that the playout buffer never overflows when the arrival pattern of stream objects at the PE is bounded by α_x and the consumption pattern of stream objects from the playout buffer is bounded by α_C.

3.2 Service Bounds Expressed in Processor Cycles

The lower and the upper bounds on the service that needs to be guaranteed by a PE, as given by inequalities (7) and (8), are specified in terms of the minimum and the maximum number of stream objects that need to be processed within any given time interval. However, due to the data-dependent variability in the execution times of multimedia tasks, the number of processor cycles required to completely process any stream object might be highly variable. As explained in Section 2, this variability can be captured by the function γ, which we refer to as the *workload curve*. It follows from the last subsection that $\gamma^u(\beta^l(\Delta))$ is the minimum number of processor cycles that must be provided to a stream within any time interval of length Δ to guarantee that the playout buffer never underflows and the internal buffer at the PE never overflows. Similarly, $\gamma^l(\beta^u(\Delta))$ is the maximum number of processor cycles that may be provided to a stream within any time interval of length Δ to guarantee that the playout buffer never overflows.

Here we would like to point out that from our definition of the function β, it follows that $\beta^l(t) \geq \beta^l(s) + \beta^l(t-s)$

Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS 2004)
1052-8725/04 $20.00 © 2004 IEEE

for all $t \geq 0$ and $0 \leq s \leq t$. Similarly, $\beta^u(t) \leq \beta^u(s) + \beta^u(t-s)$ for all $t \geq 0$ and $0 \leq s \leq t$. However, the bounds given by inequalities (7) and (8) need not satisfy these properties. Let us assume that inequality (7) is of the form $\beta^l(t) \geq f(t)$, $\forall t \geq 0$ and inequality (8) is of the form $\beta^u(t) \leq g(t)$, $\forall t \geq 0$, i.e. $f(t)$ is the right hand side term of inequality (7) and $g(t)$ is the right hand side term of inequality (8). Now let us define two functions σ^l and σ^u as follows:

$$\sigma^l(\Delta) = \begin{cases} 0 & \text{if } \Delta = 0 \\ \gamma^u(f(\Delta)) & \text{if } \Delta = 1 \\ \max\{\gamma^u(f(\Delta)), (\sigma^l \overline{\otimes} \sigma^l)(\Delta)\} & \text{if } \Delta > 1 \end{cases} \tag{9}$$

$$\sigma^u(\Delta) = \begin{cases} 0 & \text{if } \Delta = 0 \\ \gamma^l(g(\Delta)) & \text{if } \Delta = 1 \\ \min\{\gamma^l(g(\Delta)), (\sigma^u \otimes \sigma^u(\Delta)\} & \text{if } \Delta > 1 \end{cases} \tag{10}$$

The functions $\sigma^l(\Delta)$ and $\sigma^u(\Delta)$ are therefore defined over $\Delta = 0, 1, 2, \ldots$, and denote the minimum and the maximum number of processor cycles that should be provided to a stream within *any* time interval of length Δ for all the buffer overflow and underflow constraints to be satisfied. Moreover, it can be shown that these two functions satisfy the properties that any function which bounds the service provided by a PE should satisfy, i.e. $\sigma^l(t) \geq \sigma^l(s) + \sigma^l(t-s)$ for all $t \geq 0$ and $0 \leq s \leq t$ and $\sigma^u(t) \leq \sigma^u(s) + \sigma^u(t-s)$ for all $t \geq 0$ and $0 \leq s \leq t$.

3.3 Bounding the Analysis Interval

So far, our computation of the service bounds σ^l and σ^u were based on the fact that the arrival curves α_x and α_C and the workload curve γ are known for all possible time interval lengths $\Delta \geq 0$. These curves would usually be derived by simulating the processing or execution of several representative audio/video samples on a template platform architecture, as explained in Section 1. The traces collected from such a simulation—from the different parts of the platform architecture, such as the arrival pattern of stream objects in front of PE_2 in Figure 1—are then analyzed to derive the different arrival and workload curves. However, since these representative audio/video samples would always be of finite length, the curves or bounds derived from the resulting traces would also be of finite length. But the platform designed on the basis of these finite length traces might later be used to process larger audio/video samples. Hence, we would like to guarantee the buffer overflow and underflow constraints on input streams of any length (provided they satisfy the bounds dictated by the arrival and the workload curves), although the analysis and the design of the platform is based on only finite length representative inputs.

We would like to point out here that in practice the above issue will not be of major concern to any system designer. He would use sufficiently long (but finite length) representative audio/video samples in the initial simulation phase to derive the bounds (i.e. arrival and workload curves) that any input belonging to the class represented by these audio/video samples is expected to satisfy. Based on these bounds, the platform architecture in question would be designed. When such an architecture processes input streams which are longer in duration than the samples used for designing the architecture, it is assumed that the variability of the entire stream is bounded by the variability existing in the sample inputs. Such assumptions are not specific to our framework and are common whenever a system is designed based on *representative inputs* (for example, see [7]). However, it is also possible to formally address this issue within our framework, details of which may be found in [8].

3.4 Extending the Analysis to Other PEs

The framework presented so far is based on the assumption that the PE being analyzed is the last one in the path of a stream i.e. its output is directly written into the playout buffer. Now let us consider a PE, whose output is fed into another PE i.e. the next PE in the path of the stream. An example of such a PE is PE_1 in Figures 1 and 2. To derive the service bounds for this PE, let us denote the arrival curve corresponding to the arrival pattern of stream objects at the internal buffer of PE_2 as α_{x_2}. Similarly, let the arrival pattern of stream objects at the internal buffer of PE_1 be bounded by α_{x_1}, and let the size of this internal buffer be b_1. Then bounds on β^l and β^u (such as those given by inequalities (7) and (8)) for PE_1 can be calculated from α_{x_1}, α_{x_2} and b_1. However, the only constraint that needs to be satisfied in this case is that the internal buffer of PE_1 should not overflow. The resulting bounds on the service are therefore much simpler than the ones derived above, and hence we omit them here. This same scheme can be applied to other PEs in the path of the stream which are away from the playout buffer. If all of the PEs provide a service in accordance with the bounds computed for them, then it is guaranteed that none of the internal buffers in the architecture will overflow, and the playout buffer will neither overflow and nor underflow.

4 Computing Processor Frequency Ranges

Given the service bounds σ^l and σ^u for a PE, in this section we compute the discrete frequency levels or the frequency range that must be supported by the PE in order to realize these service bounds. For any given multimedia application and a class of input streams to be processed, accurately determining the appropriate processor frequency range is a non-trivial problem. A straightforward approach would be to choose a processor frequency that is sufficient to process stream objects at a rate equal to the long-term average rate at which stream objects are consumed by the output real-time client. However, since the input stream

Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS 2004)
1052-8725/04 $20.00 © 2004 **IEEE**

is bursty in nature and there is data-dependent variability in the execution requirement of stream objects, processing stream objects at this average rate might lead to buffer overflows and underflows. Therefore, the commonly followed practice is to choose a processor frequency which is slightly higher than what is required to process stream objects at the average consumption rate. However, there is no well known formal guideline on how much higher this chosen frequency should be. If the chosen frequency is much higher than what is required to counter the effects of bursts, then complicated blocking reads and writes must be used, which requires substantial operating system support.

The situation is much more complicated when the PE in question has to process multiple classes of input streams or multiple applications. In such cases, the different input classes might have different computational demands and hence require different processor frequencies. Here is it important to determine the range of processor frequencies that must be supported for each class. If these ranges overlap, then the processor might support some frequency belonging to this overlapping range. But if these ranges do not overlap, then multiple frequency levels need to be supported. Further, the processor frequency range to be supported by a PE is heavily dependent on the size of the on-chip buffers. A platform designer would therefore be interested in obtaining insights into this dependency. Our results presented below would help in obtaining such insights. These results can be summarized as follows: (i) For any application and a class of input streams, we can statically generate frequency schedules for a PE which satisfy all the buffer constraints. Such schedules specify the frequency with which the PE should be run at any time. (ii) We derive a frequency range $(f_{\min}, f_{\max})$ such that *all* feasible frequency scheduling algorithms will only use frequencies within this range. Therefore, it would be sufficient if the PE supports frequencies belonging to this range only. (iii) We also derive a frequency range $(\bar{f}_{\min}, \bar{f}_{\max})$ such that at any time, choosing any frequency within this range will lead to all the buffer constraints being satisfied. However, such frequency schedules will be a subset of *all* possible feasible frequency schedules. Further, $(\bar{f}_{\min}, \bar{f}_{\max}) \subseteq (f_{\min}, f_{\max})$. The main difference between the above two frequency ranges is that if a PE supports only the latter range, then the set of possible frequency schedules is restricted but the frequency scheduling algorithm is simpler because at any time, any frequency within this range can be selected. We call the former range the *History-Dependent Frequency (HDF) Range* since the frequency that can be chosen at any time depends on the frequencies chosen at previous time instants. The latter range is called the *History-Independent Frequency (HIF) Range*. The main motivation behind identifying the HDF range is that it allows more flexibility in choosing a frequency schedule: based on the HDF range, a designer can choose a processor which supports only a fixed number of frequency levels and runs this processor using a static frequency schedule which contains only these frequency levels. However, these frequency levels might not belong to the HIF range. (iv) Finally, our framework can also be used to identify how the bounds $(f_{\min}, f_{\max})$ and $(\bar{f}_{\min}, \bar{f}_{\max})$ change by changing the on-chip buffer sizes (both the playout and the internal buffers).

For simplicity, we assume that the processor frequency can be changed at each time unit. Then during any run of the processor over a time interval of length n, let its frequency values be $f_1, \ldots, f_n$, i.e. f_i is the frequency at which the processor is run during the time interval $t = (i-1, i]$. The service being offered to a stream as a result of this schedule has to be bounded by the service curves σ^l and σ^u. Given this constraint, the lower and upper bound on any f_i can be shown to be equal to (see [8] for details): $f_i^l = \max_{1 \le j \le i-1}\{\sigma^l(1), \sigma^l(i-j+1) - \sum_{p=j}^{i-1} f_p\}$, and $f_i^u = \min_{1 \le j \le i-1}\{\sigma^u(1), \sigma^u(i-j+1) - \sum_{p=j}^{i-1} f_p\}$. f_i^l and f_i^u depend on all the previous frequency assignments $f_1, \ldots, f_{i-1}$. To generate a static frequency schedule, we can choose any $f_i \in [f_i^l, f_i^u]$, and the chosen f_i will determine the range $[f_{i+1}^l, f_{i+1}^u]$. Now, to compute the HDF range, that was mentioned above, we first define two functions $(\sum_{p=j}^{i-1} f_p)^l$ and $(\sum_{p=j}^{i-1} f_p)^u$. The first provides a lower bound, and the second an upper bound on the sum $f_j + \ldots + f_{i-1}$.

$$(\sum_{p=j}^{i-1} f_p)^l = \max_{1 \le q \le j-1}\{\sigma^l(i-j), \sigma^l(i-q) - (\sum_{p=q}^{j-1} f_p)^u\}$$

$$(\sum_{p=j}^{i-1} f_p)^u = \min_{1 \le q \le j-1}\{\sigma^u(i-j), \sigma^u(i-q) - (\sum_{p=q}^{j-1} f_p)^l\}$$

Using the above two functions, we now define two additional functions: $f_i^{\min} = \max_{1 \le j \le i-1}\{\sigma^l(1), \sigma^l(i-j+1) - (\sum_{p=j}^{i-1} f_p)^u\}$ and $f_i^{\max} = \min_{1 \le j \le i-1}\{\sigma^u(1), \sigma^u(i-j+1) - (\sum_{p=j}^{i-1} f_p)^l\}$. $f_i^{\min}$ is the smallest possible processor frequency that can be assigned during the time interval $(i-1, i]$, and $f_i^{\max}$ is the largest possible processor frequency that can be assigned during this time interval. Then the HDF range is defined as: $f_{\min} = \min_{i=1,\ldots,n}\{f_i^{\min}\}$, and $f_{\max} = \max_{i=1,\ldots,n}\{f_i^{\max}\}$. The HIF range is defined as: $\bar{f}_{\min} = \max_{\Delta=1,\ldots n}\{\sigma^l(\Delta)/\Delta\}$, and $\bar{f}_{\max} = \min_{\Delta=1,\ldots n}\{\sigma^u(\Delta)/\Delta\}$. As mentioned above, frequency schedules restricted to this range can choose any frequency value within this range at any time instant. Details on how the above results were derived, may be found in [8].

5 Case Study: MPEG-2 Decoder Application

In this section we present a case study to illustrate the use of the framework developed in the last two sections. Towards this, we map an MPEG-2 decoder application onto a platform architecture consisting of two PEs. The goal is to compute the processor frequency range that needs to be

Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS 2004)
1052-8725/04 $20.00 © 2004 **IEEE**

supported by one of the PEs and also identify how this range changes with different on-chip buffer sizes.

As shown in Figure 1, the MPEG-2 decoder application is partitioned into a set of tasks executing in parallel on two PEs of the platform architecture. PE_1 executes the variable length decoding (VLD) and the inverse quantization (IQ) tasks, while PE_2 executes the inverse discrete cosine transform (IDCT) and the motion compensation (MC) tasks. A compressed video bit stream arrives from the network interface into the input buffer of PE_1. After being processed on PE_1, the partially decoded stream of *macroblocks* enters the buffer B_2 in front of PE_2. PE_2 reads this buffer, one macroblock at a time, and computes for each macroblock the IDCT and MC functions. Finally, the video stream emerges out of PE_2 as a fully decoded stream of macroblocks. This stream is written into the playout buffer B_v, which is read at a constant rate by the video output port V_{out}. The video output port represents the real-time client (RTC) in this setup. The rate at which it reads the playout buffer B_v is determined by the resolution and the frame rate of the decoded MPEG-2 video sequence. In the above setup, none of the buffers B_1, B_2 and B_v are ever allowed to overflow, and the playout buffer B_v should never underflow. Determining the service that must be offered by the PEs to the stream, and thereby identifying their feasible clock frequency ranges under the given buffer constraints is not an easy task, for reasons that we have discussed before.

Now, using PE_2 as our example, we will demonstrate how the methodology proposed in the last two sections can be applied to compute the required service bounds σ and the associated feasible clock frequency ranges for the given MPEG-2 decoder application.

5.1 Computing the Service Bounds and the Frequency Range for PE_2

Before we can compute the service bounds σ and the corresponding HDF and HIF ranges for PE_2, we need to obtain the arrival curves α_x and α_C, and the workload curve γ which characterize the stream processed by PE_2 (that is consumed by V_{out}). In general, there are different possibilities for obtaining these curves. In some cases it might be possible to derive these curves analytically from a formal specification of the system and its environment. In other cases, a simulation and trace-based analysis approach might be necessary and indeed sufficient for the problem at hand. In our case, we adopt the latter method to obtain the α_x and the γ curves. Towards this, we collect execution traces by simulating an *abstract model* of the platform architecture and then we analyze the obtained traces to derive the required curves. The abstract model of the platform is based on an instruction set simulator, which we use to obtain the traces of execution demands of the MPEG-2 decoder tasks. In our experimental setup we use a customized version of the SimpleScalar instruction set simulator for this purpose.

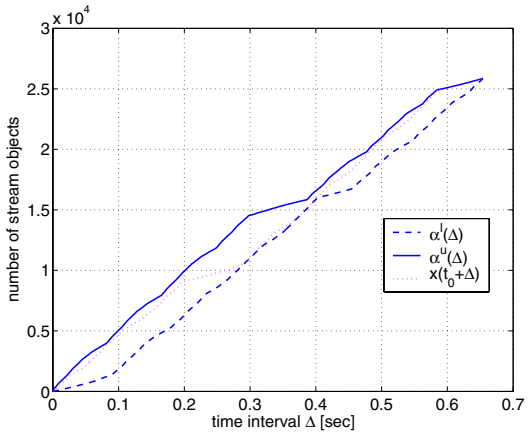

Figure 3. (α_x^l, α_x^u) and $x(t)$ corresponding to the macroblock stream (for $video_1$) at the output of PE_1.

The arrival curves (α_x^l, α_x^u) of the stream at the output of PE_1 can be obtained by measuring the execution demands of the VLD and IQ tasks for each macroblock in the video sequence and by taking into account (i) the constant arrival rate of the compressed bit stream at the input of PE_1, and (ii) the number of bits allocated to encode each macroblock in the stream. In this procedure, we first obtain the cumulative function $x(t)$ describing the arrival of macroblocks in the buffer B_2 shown in Figure 1. We then analyze it using time intervals of different lengths, to obtain the arrival curves. Figure 3 shows the arrival curves (α_x^l, α_x^u), which we have obtained by applying the above method to a representative 4 Mbps video sequence, which we refer to as $video_1$.

In a similar way, we also derived the arrival curves (α_C^l, α_C^u). However, in this case, since we precisely know the characteristics of the real-time client, we do not need to rely on simulation. α_C^l and α_C^u can be constructed analytically, using the fact that the real-time client reads macroblocks from the playout buffer at a specified constant rate.

To obtain the workload curves (γ^l, γ^u), we first collect a trace of execution demands for the pair of tasks IDCT and MC executing on PE_2. Using a method similar to the one used for obtaining the arrival curves α_x, we then analyze this trace to identify the maximum and the minimum processing demand imposed by any sequence of k consecutive macroblocks within the video sequence.

Now we apply the results presented in Section 3 to compute the cycle-based service bounds (σ^l, σ^u) corresponding to the service that must be offered by PE_2, to any video stream belonging to the class of streams bounded by the curves α_x, α_C and γ. The bounds (σ^l, σ^u) corresponding to the example video sequence $video_1$, for two different *system configurations* are shown in Figure 4. The two system configurations differ only in the sizes of the buffers B_v and B_2. By examining the plots in Figure 4, we can see that even a relatively small change in the available buffer space

Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS 2004)
1052-8725/04 $20.00 © 2004 **IEEE**

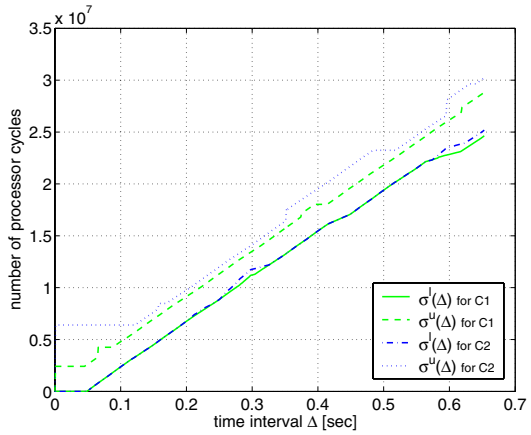

Figure 4. (σ^l, σ^u) for $video_1$ for two different system configurations $C1$ and $C2$, where $C1 = \{B_2 = 4500, B_v = 5000\}$ and $C2 = \{B_2 = 4000, B_v = 5600\}$.

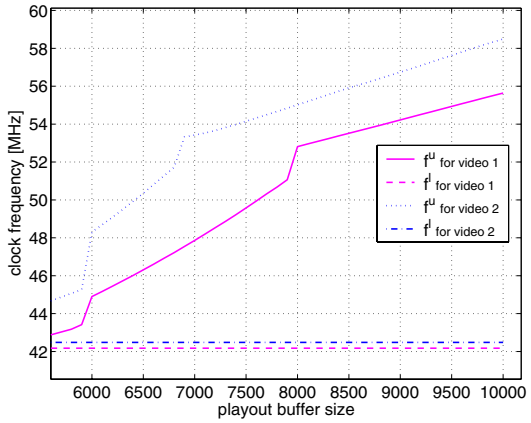

Figure 5. Dependency of HIF ranges on the playout buffer size for two different classes of the MPEG-2 video streams: 4 Mbps ($video_1$) and 8 Mbps ($video_2$). The size of buffer B_2 is fixed to 4000 macroblocks.

can have a considerable impact on the service bounds. Furthermore, the distribution of the total on-chip buffer space among the different buffers may also have an impact on the service bounds.

Video sequences belonging to different classes of streams may have very different on-chip buffer requirements. Therefore, the service bounds for these sequences, and hence their feasible clock frequency ranges might also be very different. This information on how different these ranges might be for different classes of video sequences, can be efficiently obtained from the service bounds σ, as described in Section 4. In our example, using the proposed framework we have computed the HIF ranges for two classes of video sequences characterized by different input bit stream rates, i.e. for 4 Mbps and 8 Mbps MPEG-2 streams. The resulting HIF ranges are shown in Figure 5.

Figure 5 also shows the dependency of the HIF range on the playout buffer size for a 4 Mbps and a 8 Mbps MPEG-2 video streams. In this figure it can be seen that the playout buffer size has a considerable impact on the upper frequency bound f^u. By increasing the buffer size, the maximum frequency with which PE_2 can run, also increases. This corresponds to the intuitive understanding that the larger the playout buffer size, the more bursty the incoming stream can be.

video	buffer sizes		schedule	measured backlogs	
	B_2	B_v		B_2	B_v
$video_1$	4000	5600	f^u	3335	4763
			f^l	3719	4621
			rand	3614	4844
			rand	3678	4980
	4500	6000	f^u	2680	5080
			f^l	4078	4615
			rand	3472	3472
			rand	3614	4844
$video_2$	4000	5600	f^u	2479	4525
			f^l	3971	4132
			rand	3368	4754
			rand	3502	4500
	4500	5000	f^u	2786	4330
			f^l	4004	4132
			rand	3227	4401
			rand	2818	4402

Table 1. The maximum buffer fill levels obtained by simulating a static frequency schedule for PE_2 that was derived using the proposed framework. $video_1$ and $video_2$ are 4 Mbps and 8 Mbps MPEG-2 video streams respectively.

Figure 5 shows an overlap in the HIF ranges of the two classes of video streams. This implies that for any *feasible* playout buffer size and a fixed size of B_2 (set to 4000 macroblocks), we can always find a clock frequency with which PE_2 can be run for video sequences belonging to both the input classes (i.e. 4 Mbps and 8 Mbps input rates). It may be noted here, that playout buffers only beyond a certain size are feasible—meaning that, only for playout buffers beyond this size, feasible service bounds σ exists. It may also be noted that in general such a common clock frequency for any two input classes might not exist for a single system configuration. In such cases it will be necessary to support multiple frequency ranges/values, where the frequency level at which the processor is run depends on the class to which the input belongs. Alternatively, the configuration of the system can be changed (for example by increasing buffer sizes), till the frequency ranges of two input classes overlap. When this happens, once again it would be sufficient for the processor to support a single frequency level belonging to this overlapping range. Our methodology can be used to efficiently identify such design tradeoffs in the case of configurable platform architectures.

5.2 Validation of the Analytical Bounds

To validate our framework, we simulated the platform architecture using static frequency schedules derived using the framework. Towards this, we used a detailed simulator

Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS 2004)
1052-8725/04 $20.00 © 2004 **IEEE**

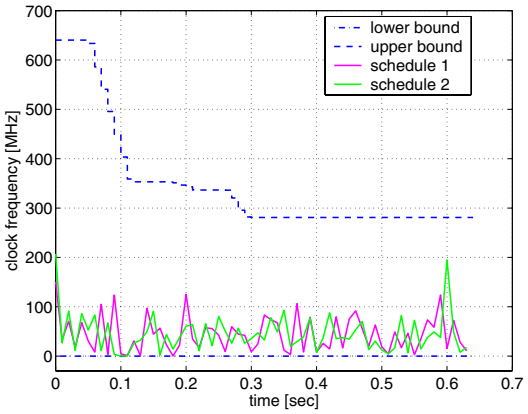

Figure 6. The HDF range and two randomly generated schedules obtained from the service bounds σ.

of the system shown in Figure 1. The simulator consisted of a transaction level model of the system architecture written in SystemC, and the models of PEs were based on a customized version of the SimpleScalar instruction set simulator. Using this simulation setup, we measured the maximum backlogs and recorded any buffer underflows that occurred as a result of running the system with a static frequency schedule for PE_2 (which was generated using our framework). The results of this simulation are summarized in Table 1.

We evaluated several frequency schedules for PE_2, that are bounded by the computed HDF and HIF ranges obtained using the proposed framework. These ranges correspond to different system configurations and classes of the video streams. In Table 1 these schedules are indicated as f^u, f^l, and $rand$. f^u and f^l denote cases where PE_2 was run with a single clock frequency determined by the upper $\bar{f}_{max}$ and the lower $\bar{f}_{min}$ HIF bounds respectively (see Figure 5 for a reference). $rand$ denotes randomly generated static schedules that satisfy the HDF bounds (f_{min}, f_{max}), as explained in Section 4. Two such randomly generated schedules are shown in Figure 6.

In all the simulations we performed, the maximum backlogs measured in the buffers never exceeded the buffer sizes. Furthermore, our simulation results also showed that playout buffer underflows never occurred for any of the simulated frequency schedules. These simulations therefore validate the proposed framework and suggest its practicality. Finally, we would once again like to point out that obtaining equivalent results using purely simulation based approaches is extremely time consuming and such approaches usually fail to provide any formal performance guarantees.

6 Concluding Remarks

Our framework can be used for the design space exploration of parameters or configurations of SoC platform architectures for multimedia processing, that contain processor cores supporting dynamic voltage/frequency scaling. In contrast to simulation based approaches, which usually follow a trial-and-error approach and involve very high simulation times, the proposed framework can provide useful insights into the design space and can aid a system designer in systematically tuning a platform architecture for a class of multimedia applications.

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".

References

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

[2] M. Buss, T. Givargis, and N. Dutt. Exploring efficient operating points for voltage scaled embedded processor cores. In *24th IEEE Real-Time Systems Symposium (RTSS)*, 2003.

[3] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *DATE*, 2003.

[4] 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.

[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, 2001.

[6] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. Srivastava. Power optimization of variable-voltage core-based systems. *IEEE Trans. on Computer Aided-Design of Integrated Circuits and Systems*, 18(12), 1999.

[7] 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.

[8] Y. Liu, A. Maxiaguine, S. Chakraborty, and W. Ooi. Processor frequency selection for SoC platforms for multimedia applications. NUS, School of Computing Technical Report TRC8/04, August, 2004, http://www-appn.comp.nus.edu.sg/~esubmit/search/.

[9] A. Maxiaguine, S. Künzli, S. Chakraborty, and L. Thiele. Rate analysis for streaming applications with on-chip buffer constraints. In *ASP-DAC*, 2004.

[10] M. Rutten, J. van Eijndhoven, E. Jaspers, P. van der Wolf, O. Gangwal, and A. Timmer. A heterogeneous multiprocessor architecture for flexible media processing. *IEEE Design & Test of Computers*, 19(4):39–50, 2002.

[11] M. Rutten, J. van Eijndhoven, and E.-J. Pol. Design of multitasking coprocessor control for eclipse. In *CODES*, 2002.

[12] M. Rutten, J. van Eijndhoven, and E.-J. Pol. Robust media processing in a flexible and cost-effective network of multitasking coprocessors. In *ECRTS*, 2002.

[13] K. Sekar, K. Lahiri, and S. Dey. Dynamic platform management for configurable platform-based system-on-chips. In *ICCAD*, 2003.

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

Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS 2004)
1052-8725/04 $20.00 © 2004 **IEEE**