Tuning SoC Platforms for Multimedia Processing: Identifying Limits and Tradeoffs

Alexander Maxiaguine[1] Yongxin Zhu[2] Samarjit Chakraborty[2] Weng-Fai Wong[2]
[1]Computer Engineering and Networks Laboratory, ETH Zürich
[2]Department of Computer Science, National University of Singapore
maxiagui@tik.ee.ethz.ch, {zhuyx,samarjit,wongwf}@comp.nus.edu.sg

ABSTRACT

We present a analytical framework to identify the tradeoffs and performance impacts associated with different SoC platform configurations in the specific context of implementing multimedia applications. "Configurations" in this case might include sizes of different on-chip buffers and scheduling mechanisms (or associated parameters) implemented on the different processing elements of the platform. Identifying such tradeoffs is difficult because of the bursty nature of on-chip traffic arising out of multimedia processing and the high variability in their execution requirements, which result in a highly irregular design space. We show that this irregularity in the design space can be precisely captured using an abstraction called *variability characterization curves*.

Categories and Subject Descriptors

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

General Terms

Algorithms, Performance, Design

Keywords

System-on-chip, multimedia systems, platform management

1. INTRODUCTION

Of late, there has been a considerable interest in generic and configurable System-on-Chip (SoC) platforms specifically targeted towards implementing multimedia applications. Examples of these are the Eclipse architecture template from Philips [9], and the Viper SoC architecture [2] which targets advanced set-top box and DTV markets. Designs based on such generic platforms are associated with flexibility, low design costs and time-to-market advantages. However, all of these come at the cost of a large disparity in performance between generic platform based designs and fully customized solutions based on ASICs. As a result, a lot of effort is currently being directed towards devising platform configuration and management techniques for narrowing this gap.

CODES+ISSS'04, September 8–10, 2004, Stockholm, Sweden.

Customizing or tuning a generic platform for a particular application at hand consists of two phases: (i) determining the optimal hardware configuration of the platform, such as sizes of on-chip buffers, bus widths, and cache configurations, and (ii) determining optimal platform management policies for the hardware configuration chosen, such as the scheduling mechanisms to be used for the different buses and processors. Typically these two phases are very tightly coupled together and are associated with several design tradeoffs. A common example of this in the context of multimedia stream processing would be: with small on-chip buffers, schedulers must allow a high *degree of preemption* to satisfy the real-time constraints associated with bursty streams. A "high degree of preemption" in the case of round robin or time division multiple access (TDMA) schedulers translate to small *periods* or *slot sizes*, which increase scheduling overheads, but lead to savings on on-chip buffer requirements. Since on-chip buffers are available only at a premium because of their high area requirements [10], a designer will be interested in identifying how the buffer requirements change with changing the scheduling mechanisms implemented on the different processors—based on which he can choose an appropriate tradeoff.

However, identifying such tradeoffs is not straightforward. Firstly, the underlying *design space* is typically very large, even in a simple case where all the schedulers to be implemented on the different processors are restricted to only TDMA schedulers and the problem is to identify how on-chip buffer requirements change with changing the *periods* of these schedulers. Secondly, on-chip traffic arising out of multimedia processing on multiprocessor architectures tends to be very complex and bursty in nature [10], which results in a very irregular design space. Most of the current efforts towards identifying such design tradeoffs and limits to performance enhancements, rely on simulation-oriented techniques. Notable among these, in the specific context of multimedia processing is the Atremis project [8], which proposes trace-driven co-simulation and symbolic execution of applications [11]. The Spade methodology and toolset [5] developed within this project can be used for fast design space exploration of heterogeneous media processors.

In contrast to this line of work, in this paper we present a analytical framework to efficiently identify the tradeoffs associated with different hardware configurations and platform management policies while implementing a multimedia application on a generic SoC platform. This framework can therefore guide a system designer to optimally tune or configure a SoC platform for any given multimedia application.

Basic idea: Our framework relies on a one-time initial co-simulation of the multimedia application to be implemented, on an *abstract model* of the platform architecture. This results in a set of traces from which certain parameters—such as the variability in the execution requirements of the application—are computed. These parameters then serve as inputs to analytical models of *concrete* instances of the platform architecture and platform management schemes. Such concrete instances have specified on-chip buffer sizes, bus widths, cache configurations, etc. and also specified platform management policies with concrete parameters (such as the period of a TDMA scheduler). The framework then returns performance metrics such as the maximum allowed input stream rates, or metrics related to the quality of the audio/video output. Alternatively, given certain constrains on the output audio/video quality required and a hardware configuration of the platform, the framework can return the set of feasible platform management schemes (such as scheduling policies along with their associated parameters). In this entire process, the initial co-simulation to generate traces needs to be done only once and subsequently any concrete instance of a platform architecture can be analyzed using only analytical means, resulting in the possibility of fast design space exploration.

This basic scheme of using an initial co-simulation to generate traces is not new and is also followed in [8]. However, simulation-oriented methods such as [5] and [11] then rely on a symbolic simulation of these traces (see also [4] for work on performance analysis of bus-based SoC communication architectures), whereas we rely on purely analytical methods which are specific to multimedia processing. Another feature of our scheme is the possibility of avoiding the initial co-simulation to generate traces from which the parameters of the analytical models are derived. Instead, these parameters can also be analytically derived from models of the application to be implemented along with models of the platform architecture, by resorting to techniques such as program path analysis and microarchitecture modeling. Some preliminary ideas along these lines have been recently proposed in [3, 12] and there are considerable research opportunities in this direction. However, in this paper we do not explore this direction, and instead rely on simulations to obtain the parameters for our models, as discussed above.

Our results and relation to previous work: The framework presented here is based on a novel concept of *variability characterization curves* (VCCs), which we develop in this paper. As mentioned above, the main complexity of our problem stems from the complex and bursty nature of on-chip traffic resulting from multimedia processing on a multiprocessor architecture. This can be attributed to two main characteristics of multimedia applications: (i) high data-dependent variability in their execution time requirements (because the execution requirement of a multimedia stream depends on the properties of the audio/video sample being processed), and (ii) variability in the input-output rates associated with multimedia processing tasks (for example, the *variable length decoding* (VLD) task in a MPEG-2 decoder requires a variable number of compressed bits from a coded bitstream to generate one *macroblock*).

The concept of VCCs is motivated by the work recently reported in [6] and [7]. These two papers proposed a worst-case characterization of multimedia streams in terms of the burstiness in their arrival patterns and the variability in their execution requirements, based on the theory of *Network Calculus* [1]. The main contribution of this paper is a generalization of the concepts proposed in [6] and [7]. Instead of resorting to seemingly different schemes for representing different characteristics, we show that all aspects of on-chip traffic arising out of multimedia processing can be characterized by a set of VCCs. Such a characterization seems to be expressive enough for the purpose of system-level design and analysis of stream processing architectures. VCCs can be used to represent not only the burstiness in the arrival pattern of streams and the variability in their execution requirements, but also the variability in the input-output rates of multimedia processing tasks and possibly other characteristics as well. Because of their inability to represent characteristics like variability in the input-output rates of tasks such as VLD, the models presented in [6] and [7] can only be used to accurately analyze restricted types of multimedia tasks. The work presented in this paper removes this restriction and allows for the analysis of general multimedia tasks implemented on a multiprocessor architecture.

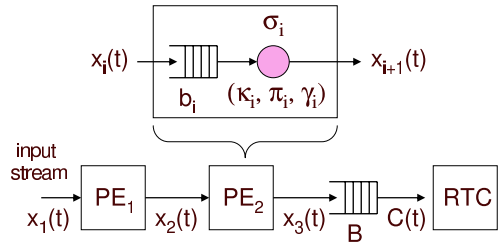

Figure 1: Structural view of multimedia processing on a multiprocessor SoC platform.

Organization of the paper: In the next section we introduce the concept of VCCs and show how they can be used to represent different characteristics of on-chip traffic resulting out of multimedia processing. In Section 3, we then demonstrate an application of VCCs by developing a framework to identify the tradeoffs between on-chip buffer requirements and the scheduling overheads associated with different schedulers to be implemented on a SoC platform. This is only one possible illustration of using VCCs in the system-level design of multimedia processors. Other design tradeoffs can as well be analytically derived in a similar fashion. Finally, in Section 4 we present an illustrative case study to show the utility of the framework developed in Section 3.

2. CHARACTERIZING VARIABILITY

We consider the following system-level view of multimedia stream processing on a SoC platform (see Figure 1). The platform architecture consists of multiple processing elements (PEs) onto which the different parts of an application are mapped. An input multimedia stream enters a PE, gets processed by the task(s) implemented on this PE, and the processed stream enters another PE for further processing. Each PE has an internal buffer, which is a FIFO channel of fixed capacity, and is used to store the incoming stream to be processed. Finally, the fully processed stream is written into a *playout buffer* which is read by some *real-time client* (RTC) such as an audio or a video output device.

For the sake of generality, we consider any multimedia stream to be made up of a sequence of *stream objects*. A stream object might be a bit belonging to a compressed bitstream representing a coded video clip, or a macroblock, or

a video frame, or an audio sample—depending on where in the architecture the stream exists. For example, if the architecture shown in Figure 1 is used to implement an MPEG-2 decoder, then stream objects belonging to the input stream might be single bits. But stream objects belonging to the stream entering the RTC might be decoded macroblocks.

VCCs are used to quantify best-case and worst-case characteristics of *sequences*. These can be sequences of consecutive stream objects belonging to a stream, sequences of consecutive executions of a task implemented on a PE while processing a stream, or sequences of consecutive time intervals of some specified length. A VCC $\mathcal{V}$ is composed of a tuple $(\mathcal{V}^l(k), \mathcal{V}^u(k))$. Both these functions take an integer k as the input parameter, which represents the *length* of a sequence. $\mathcal{V}^l(k)$ then returns a *lower bound* on some property that holds for *all* subsequences of length k within some larger sequence. Similarly, $\mathcal{V}^u(k)$ returns the corresponding *upper bound* that holds for *all* subsequences of length k within the larger sequence. Let the function P be a *measure* of some property over a sequence $1, 2, \ldots$. If $P(n)$ denotes the measure of this property for the first n items of the sequence (i.e. $1, \ldots, n$), then $\mathcal{V}^l(k) \leq P(i+k) - P(i) \leq \mathcal{V}^u(k)$ for all $i, k \geq 1$. As examples, let us now consider the following different realizations of a VCC.

Workload curve $\gamma = (\gamma^l, \gamma^u)$: A stream is to be processed by a task T implemented on a PE. The execution requirements of T for processing different stream objects belonging to this stream is variable, and we would like to use a VCC γ to quantify this variability. We use $\gamma^l(k)$ to denote the minimum number of executions of T that is guaranteed to be completed by a sequence of k processor cycles of the PE. $\gamma^u(k)$ is the maximum number of executions of T that may be completed using k processor cycles.

Now consider a sequence of executions $1, 2, 3, \ldots$ of T. Let the function $W(n)$ denote the total number of processor cycles required to complete the first n executions of T, i.e. executions $1, 2, \ldots, n$. Then from the above definition of γ, the following inequalities hold: $W(i + \gamma^l(k)) - W(i) \leq k,\ \forall k, i > 0$ and $W(i + \gamma^u(k)) - W(i) \geq k,\ \forall k, i > 0$.

Let $e_{\max}$ and $e_{\min}$ be the maximum and the minimum number of processor cycles required by any single execution of T. Then typically $\gamma^l(k)$ would be greater than $\lfloor k/e_{\max} \rfloor$ and $\gamma^u(k)$ would be less than $\lfloor k/e_{\min} \rfloor$. This is because the execution requirement of the stream, as mentioned above, is variable and it is unlikely that many consecutive stream objects will all require the same number of processor cycles when being processed by T. Hence, the VCC γ is more expressive compared to simple best- or worst-case characterizations commonly used in the real-time systems domain.

It is also meaningful to construct a *pseudo-inverse* of a VCC $\mathcal{V}$, which we denote as $\mathcal{V}^{-1}$. In the case of a workload curve, $\gamma^{l^{-1}}(k)$ again takes an integer k as an argument and returns the minimum number of processor cycles that will be required by k consecutive executions of the task T. Similarly, $\gamma^{u^{-1}}(k)$ returns the maximum number of processor cycles that can be required by k consecutive executions of T. Therefore, $\gamma^{l^{-1}}(k)$ and $\gamma^{u^{-1}}(k)$ represent lower and upper bounds respectively on the execution requirements of *any* k consecutive activations of the task T.

Both the VCCs, γ and γ^{-1}, can be computed from a trace of processor cycle consumption due to a sequence of activations of a task. Such a trace may be derived from an initial co-simulation of the application and the platform architecture using representative audio/video clips, as explained in Section 1. This is also explained further in Section 4.

Consumption and production curves $\kappa = (\kappa^l, \kappa^u)$ and $\pi = (\pi^l, \pi^u)$: Let an input stream processed by a task T result in an output stream. Each activation of T consumes a variable number of stream objects belonging to the input stream, and results in the production of a variable number of output stream objects, possibly of a different type. This variability in the consumption and production rates of T can be quantified using two VCCs κ and π, which we refer to as the consumption and the production curves respectively. $\kappa^l(k)$ takes an integer k as an argument and returns the minimum number of activations of T that will be required to completely process any k consecutive stream objects. Similarly, $\kappa^u(k)$ returns the maximum number of activations of T that might be required to process any k consecutive stream objects.

We define $\pi^l(k)$ to be the minimum number of stream objects guaranteed to be produced due to any k consecutive activations of T. Similarly, $\pi^u(k)$ is the maximum number of stream objects that can be produced due to any k consecutive activations of T. Therefore, k consecutive stream objects at the input of T will result in at least $\pi^l(\kappa^l(k))$ and at most $\pi^u(\kappa^u(k))$ stream objects at its output.

Service curve $\sigma = (\sigma^l, \sigma^u)$: This VCC is used to express the variability in the processor availability. Given a PE and a stream to be processed on it, $\sigma^l(\Delta)$ is the minimum number of processor cycles that is guaranteed to be available to the stream within *any* time interval of length Δ. $\sigma^u(\Delta)$ is the maximum number of processor cycles that may be available to the stream within any time interval of length Δ.

3. IDENTIFYING DESIGN TRADEOFFS

In this section we demonstrate one possible application of VCCs in the system-level design of media processing platforms, by developing an analytical framework to identify the tradeoffs between scheduler overheads and on-chip buffer requirements.

Let an application be partitioned and mapped onto the two processors PE_1 and PE_2 shown in Figure 1. The input stream, say s, to be processed by this application is represented by a function $x_1(t)$, which denotes the number of stream objects arriving at PE_1 during the time interval $[0, t]$. After being processed on PE_1, the resulting stream, which is represented by a similar function $x_2(t)$, is processed on PE_2 and the final processed stream represented by $x_3(t)$ is written into a playout buffer of size B. This buffer is read by the RTC at a rate specified by the function $C(t)$, which denotes the number of stream objects read during the time interval $[0, t]$.

On each of PE_1 and PE_2, the stream s is scheduled by a scheduler, which apart from s also schedules other streams and real-time tasks possibly implemented on these processors. The *service* received by s at the two processors can be specified by service curves σ_1 and σ_2, where these functions depend on the scheduling policies used and their associated parameters. The variability associated with processing s on the two processors is specified by the VCCs $(\kappa_1, \pi_1, \gamma_1)$ and $(\kappa_2, \pi_2, \gamma_2)$.

Given these VCCs and the functions $x_1(t)$ and $C(t)$, for any set of buffer sizes b_1, b_2 and B we are only interested

in the set of schedulers for which no buffer overflows and the playout buffer does not underflow. The playout buffer underflow constraint is to ensure that the output quality of the audio/video stream, as specified by the function $C(t)$, is guaranteed. Clearly, the stream s should be served at both PE_1 and PE_2 at an average rate which is equal to the long-term average consumption rate of the RTC, and this should also be equal to the long-term average arrival rate of the stream at PE_1. If these average rates are not equal, then a buffer overflow or underflow is bound to occur at some point in time irrespective of the sizes of the buffers are. Although this condition on the average rate of service constraints the set of feasible schedulers, the feasible set might still be very large. Moreover different schedulers from this set might be associated with different scheduling overheads.

For example, let both PE_1 and PE_2 use TDMA schedulers with *periods* equal to p_1 and p_2 respectively and the weights associated with the stream s be w_1 and w_2 ($w_1, w_2 \leq 1$). Therefore, within a period of length p_i, s receives $w_i \times p_i$ ($i = 1, 2$) units of PE_i's processor time. Clearly, w_1 and w_2 should be chosen such that s is served at an average rate equal to the average consumption rate of the RTC. However, there is flexibility in choosing p_1 and p_2. For some values of these periods, there might be no buffer sizes which can guarantee the buffer overflow and underflow constraints. This is because of the burstiness and the variability in the execution requirements of the stream on the two processors.

From the set of feasible values of these periods (i.e. for which there exists finite buffer sizes which guarantee the overflow and underflow constraints), the smaller the length of the periods, smaller will be the on-chip buffer requirements, but at the cost of higher scheduling overheads. The goal here is therefore to identify this *tradeoff curve* between scheduling overhead and on-chip buffer requirements for different values of p_1 and p_2.

On-chip buffer requirements as a function of scheduler parameters: As shown in Figure 1, our representative platform architecture consists of multiple PEs connected by FIFO channels. Note that any scheduler implemented on a PE can be specified in terms of the service it provides to the different streams being processed, using the service curve σ. Given a stream and the service σ provided to it on each of the PEs of the platform architecture, we would like compute whether there exists finite buffer sizes which guarantee the buffer overflow and underflow constraints mentioned above. When such buffer sizes exist, we would like to compute their minimum values for which the buffer overflow and underflow constraints will be satisfied.

Let us consider the i-th PE in the path of a stream (for simplicity Figure 1 is shown with only two PEs). The arrival pattern of the stream at the input of this PE is specified by the function $x_i(t)$ and the processed stream is specified by $x_{i+1}(t)$, which serves as the input to the next PE. We use $x_i^{\min}$ and $x_i^{\max}$ to denote lower and upper bounds on x_i (i.e. $x_i^{\min}(t) \leq x_i(t) \leq x_i^{\max}(t),\ \forall t \geq 0$). We refer to the task processing the stream on PE_i as task T.

During the time interval $[0, t]$, the minimum and maximum number of activations of T that can be *requested* by the stream is equal to $\kappa_i^l(x_i^{\min}(t))$ and $\kappa_i^u(x_i^{\max}(t))$ respectively. Since the service guaranteed to the stream on PE_i is σ_i, the minimum and maximum number of activations of T that are *possible* during $[0, t]$ are $\gamma_i^l(\sigma_i^l(t))$ and $\gamma_i^u(\sigma_i^u(t))$ respectively. Therefore, the minimum and the maximum number of activations of T that *occur* in $[0, t]$, which we denote using $a_{\min}(t)$ and $a_{\max}(t)$ respectively, can be computed as follows (see [1] for the mathematical background). $a_{\min}(t) = \kappa_i^l(x_i^{\min}) \otimes \gamma_i^l(\sigma_i^l)(t)$ and $a_{\max}(t) = \kappa_i^u(x_i^{\max}) \otimes \gamma_i^u(\sigma_i^u)(t)$ where, for any two functions f and g, the min-plus convolution of f and g is given by $(f \otimes g)(t) = \inf_{s:0\leq s\leq t}\{f(t-s) + g(s)\}$. Then clearly, $x_{i+1}^{\min} = \pi_i^l(a_{\min}(t))$ and $x_{i+1}^{\max} = \pi_i^u(a_{\max}(t))$. x_{i+1} serves as the input to PE_{i+1} and is possibly composed of stream objects of a different type, compared to the stream objects at the input of PE_i. The above bounds can be used to compute the maximum backlog of stream objects at the input of PE_i. This backlog denotes the minimum buffer size that must be provided in PE_i to guarantee that an overflow does not occur. Hence, the buffer size b_i (see Figure 1) is given as follows (which can be derived from [1]):

$$b_i = \sup_{t\geq 0}\{x_i^{max}(t) - \kappa_i^{l\,-1}(a_{\min}(t)), x_i^{max}(t) - \kappa_i^{u\,-1}(a_{\max}(t))\}$$

When x_i represents the stream being written into the playout buffer, then the minimum size of the playout buffer is given by $B = \sup_{t\geq 0}\{x_i^{max}(t) - C(t)\}$. It may be noted that the inequality $x_i^{\min} \geq C(t)$ should hold for all $t \geq 0$ to guarantee that the playout buffer never underflows. If this inequality fails to hold for some value of t, then the chosen σs do not have any corresponding feasible buffer sizes. The sum of all the buffer sizes b_i belonging to the PEs in the path of the stream, along with the playout buffer size B, is the minimum buffer requirement to guarantee that no overflows occur for the chosen schedulers (specified by the σ_is).

4. ILLUSTRATIVE CASE STUDY

In this section we present a case study to illustrate the analytical framework developed in the previous section. Our platform architecture consists of two PEs onto which an MPEG-2 decoder application is partitioned and mapped. Both the PEs use TDMA schedulers to schedule the input video stream along with possibly other streams and real-time tasks. The goal is to analytically derive the *tradeoff curve* showing how the on-chip buffer requirements associated with the video stream being decoded change with different periods of the TDMA schedulers.

The high-level model of the platform architecture is captured in the structural view shown in Figure 1, which has been described in Section 2. In our setup the MPEG-2 decoder algorithm is implemented on the two PEs, PE_1 and PE_2, with each PE executing a part of the algorithm. PE_1 reads a compressed video stream from the internal buffer associated with it and performs the *variable length decoding* (VLD) and *inverse quantization* (IQ) tasks on it. PE_2 works on the partially decoded data produced by PE_1. It computes the *inverse discrete cosine transform* (IDCT) and, if necessary, applies the *motion compensation* (MC) function. After the processing on PE_2, decoded video samples are sent to a video output port (RTC), which finalizes the processing and sends the video stream to a display device.

As we explained in the preceding sections, the exchange of stream objects in the architecture occurs via FIFO buffers. In our case, the input buffer in front of PE_1 stores *bits* of the compressed video stream to be decoded; the buffer in front of PE_2 stores partially decoded *macroblocks*, and the playout buffer stores *macroblocks* of completely decoded video samples. These determine the granularity of the stream objects in the different parts of the processing chain, which we

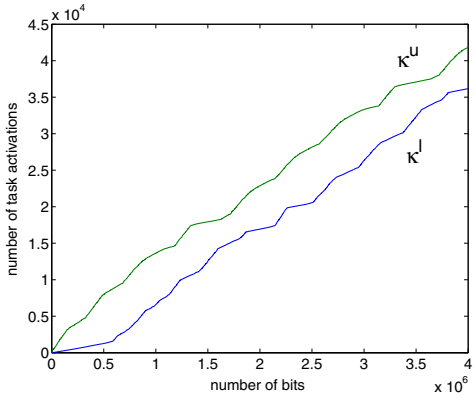

Figure 2: Consumption curve $\kappa_1 = (\kappa_1^l, \kappa_1^u)$ of the VLD+IQ task executed on PE_1.

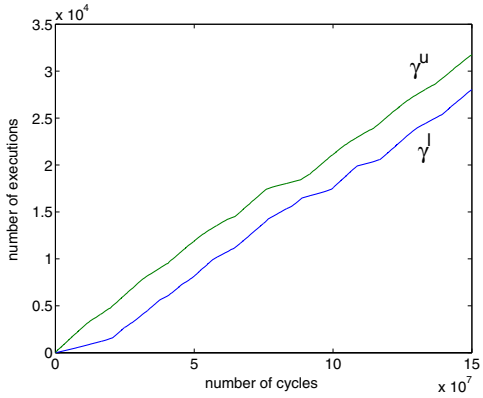

Figure 3: Workload curve $\gamma_1 = (\gamma_1^l, \gamma_1^u)$ of the VLD+IQ task executed on PE_1 .

use to define the input function $x_1(t)$, the output function $C(t)$ and the VCCs $(\kappa_i, \pi_i, \gamma_i)$ for the tasks executing on PE_1 and PE_2.

For a constant bit rate input video stream, which we use here as an example, $x_1(t) = r_c t$, where r_c denotes the bit rate of the compressed video stream at the input of the system (i.e. at the input of PE_1). Similarly, $C(t) = r_{mb}t + \tau$, where r_{mb} denotes the macroblock consumption rate, which is determined by the frame rate and the resolution of the decoded video clip, and τ denotes the initial buffering delay. In our setup $r_c = 4{\cdot}10^6$ bits/sec, $r_{mb} = 39600$ macroblocks/sec and $\tau = 0.24$ sec.

The curves π_1, π_2 and κ_2 are straight lines with slopes which correspond to the constant-rate production (or consumption) of one stream object per task activation. In contrast to this, the curve κ_1 has a complex form, since PE_1 consumes a variable number of bits from the input buffer per activation of the VLD+IQ task. Figure 2 shows κ_1, corresponding to an MPEG-2 video sequence which we used in our experiments. This curve was obtained by analyzing a trace generated from the initial co-simulation step described in Section 1. In this case study, this step comprised of simulating the execution of PE_1 and PE_2 for a representative MPEG-2 video clip using the SimpleScalar instruction set simulator. For a purely simulation-based evaluation of a platform architecture, execution traces collected from the SimpleScalar simulation were then used as input to a transaction-level model of the platform architecture in SystemC. In the analytical framework, the execution traces collected from the SimpleScalar simulation were analyzed to derive the various VCCs, thereby avoiding the SystemC-based simulation. Both the MPEG-2 decoding tasks running on PE_1 and PE_2 have variable execution times. The corresponding workload curves γ_1 and γ_2 capture this variability. As mentioned above, these curves were obtained by analyzing traces of the task execution requirements collected using the SimpleScalar instruction set simulator. For example, the γ_1 that we use as an input to our analytical model is shown in Figure 3.

To allow for processing of tasks other than the MPEG-2 decoder, PE_1 and PE_2 are scheduled using two TDMA schedulers. The VCCs σ_1 and σ_2, which characterize these schedulers have the form of periodic staircase functions [1]. Due to space restrictions we do not plot them here. However, we point out that the shapes of σ_1 and σ_2 depend on the values of the TDMA periods (or slot sizes) p_1 and p_2 respectively.

The functions $x_1(t)$, $C(t)$ and the VCCs $(\kappa_i, \pi_i, \gamma_i)$ and σ_i constitute the full specification of the problem that we want solve using the analytical framework presented in Section 3. Here, we once again point out that although obtaining this specification requires the simulation of the MPEG-2 decoder tasks, we need to do it only once, using a representative video clip (or a set of clips). Once this specification is obtained, multiple instances of the platform architecture (with different configurations) can be analyzed using only analytical means. Furthermore, we can avoid a time-consuming simulation of the whole multiprocessor system—rather, we simulate an *abstract model* of the platform for which we need to employ only an instruction set simulator.

Now, given this problem specification we can compute the maximum backlogs in the FIFO buffers of the system and check the underflow condition in the playout buffer (as described in Section 3). Here, we are interested in studying how the amount of buffer space required for processing any MPEG-2 video stream depends on the granularity of the TDMA schedulers implemented on PE_1 and PE_2. Hence, an instance of the platform configuration is determined by the pair of the TDMA periods (p_1, p_2). For each pair (p_1, p_2) we iteratively compute the maximum backlogs in the FIFO buffers and scale the obtained values by the maximum size (in bits) of the stream objects associated with the buffers. The results of this computation for a sample MPEG-2 video sequence are presented in Figure 4. These computations were carried out using a combination of Mathematica and Matlab tools in which we implemented the results derived in Section 3.

By inspecting the 3D surface shown in Figure 4 we can see the expected trend: decreasing the values of the TDMA periods p_1 and p_2, in general, lead to a reduction in the on-chip memory requirements (which implement the buffers). However, we also can see that this reduction is not uniform across the entire range of values of the periods. Even for a simple scheduling discipline like TDMA, there are large irregularities in the design space. This makes it virtually impossible to come up with an appropriate tradeoff, based only on a designer's experience on how on-chip memory requirements typically change with small changes in the parameters of the schedulers. However, since on-chip buffers have large area requirements, such an information about the design space is essential for determining optimal platform management policies. Using our framework it is therefore possible to discover the irregularities in the design space, and from it arrive at an appropriate tradeoff—in this case between scheduling

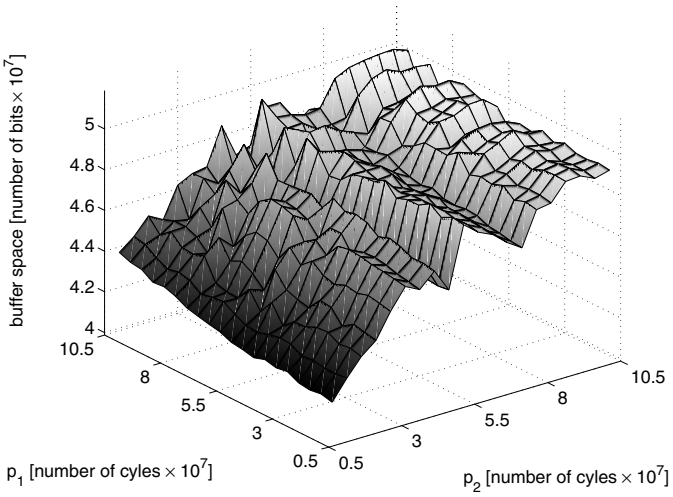

Figure 4: The surface showing the dependency of memory requirements on the values of the periods p_1 and p_2.

overheads and buffer requirements. This capability of the framework can be attributed to the underlying concept of VCCs which can be used to precisely represent the different types of variabilities associated with multimedia processing on multiprocessor SoC platforms.

Besides maximum buffer requirements, a system designer might also be interested to know whether a given choice of schedulers might lead to an underflow of the playout buffer. Our framework can also be used to answer this question. As an example, Figure 5 shows the different combinations of the periods p_1 and p_2 that might lead to a playout buffer underflow. The two axes show the different values of p_1 and p_2 and the white cells correspond to combinations of p_1 and p_2 for which there might be a playout buffer underflow. For ease of understanding, this figure can be interpreted as the horizontal plane of the 3D axes shown in Figure 4.

To validate our analytical framework, we simulated architectures corresponding to a subset of points (p_1, p_2) from Figure 4, and measured the maximum buffer fill levels. As mentioned before, this simulation was based on an abstract transaction-level model of the platform architecture in SystemC. Our analytical results always had close correspondence with the results obtained from simulation. For each platform configuration (p_1, p_2), the SystemC-based simulation required almost an hour of simulation time, which was around 100 times slower than the Mathematica + Matlab-based implementation of the analytical model. We believe that an efficient C/C++ implementation would be at least 5-10 times faster than our current prototype Mathematica + Matlab implementation. It may be noted that, considering the time involved in simulating even a single point (p_1, p_2) (for a 2 sec long video clip), it is almost infeasible to obtain a *design surface* such as the one shown in Figure 4 using purely simulation-based techniques.

5. CONCLUDING REMARKS

In this paper we introduced a novel abstraction called *variability characterization curves* to capture different properties of on-chip traffic resulting out of multimedia processing on a multiprocessor SoC platform. Using this abstraction, it is possible to precisely capture the irregularities in a design space using analytical methods, which leads to

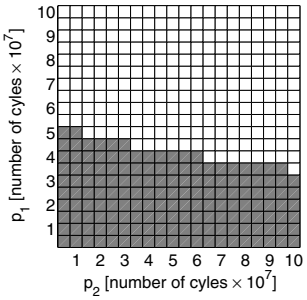

Figure 5: Scheduler periods (p_1, p_2) which might lead to playout buffer underflows (white cells).

the possibility of fast design space exploration of hardware-software architectures of media processing platforms. Standard methods for worst/best-case analysis from the real-time systems area lack sufficient expressiveness to capture different forms of variability associated with on-chip multimedia processing. Therefore, using these methods it would not be possible to derive a *design surface* as varied as the one shown in Figure 4.

Acknowledgments: This work is partially supported by the NUS URC grant R-252-000-190-112.

6. REFERENCES

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

[2] S. Dutta, R. Jensen, and A. Rieckmann. Viper: A multiprocessor SOC for advanced set-top box and digital TV systems. *IEEE Design & Test of Computers*, 18(5):21–31, September-October 2001.

[3] M. Jersak, R. Henia, and R. Ernst. Context-aware performance analysis for efficient embedded system design. In *DATE*, 2004.

[4] 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.

[5] P. Lieverse, T. Stefanov, P. van der Wolf, and E.F. Deprettere. System level design with Spade: an M-JPEG case study. In *ICCAD*, 2001.

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

[7] A. Maxiaguine, S. Künzli, and L. Thiele. Workload characterization model for tasks with variable execution demand. In *7th DATE*, March 2004.

[8] A.D. Pimentel, L.O. Hertzberger, P. Lieverse, P. van der Wolf, and E.F. Deprettere. Exploring embedded-systems architectures with Artemis. *IEEE Computer*, 34(11):57–63, 2001.

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

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

[11] V.D. Živković, E.A. de Kock, E.F. Deprettere, and P. van der Wolf. Fast and accurate multiprocessor architecture exploration with symbolic programs. In *6th DATE*, 2003.

[12] E. Wandeler, A. Maxiaguine, and L. Thiele. Quantitative characterization of event streams in analysis of hard real-time applications. In *10th IEEE RTAS*, 2004.