Reducing Data-Memory Footprint of Multimedia Applications by Delay Redistribution

Balaji Raman[1] Samarjit Chakraborty[1] Wei Tsang Ooi[1] Santanu Dutta[2]
[1]Department of Computer Science, National University of Singapore
[2]nVIDIA Corporation, Santa Clara
{ramanbal,samarjit,ooiwt}@comp.nus.edu.sg, sdutta@nvidia.com

ABSTRACT

It is now common for multimedia applications to be partitioned and mapped onto multiple processing elements of a system-on-chip architecture. An important design constraint in such architectures is that the FIFO buffers connecting the processing elements (in a pipelined fashion) should not overflow and the playout buffer should never underflow. To meet these constraints, an usual design practice is to increase the initial playout delay after which the output device starts reading from the playout buffer. Although implementing this technique is straightforward and involves only the the computation of an appropriate playout delay, it suffers from the downside of a large playout buffer being required. In this paper, instead of associating the playout delay solely with the output device, we propose to redistribute this delay among all the processing elements running the various tasks of the multimedia application. We show that this delay redistribution technique can signficantly reduce (up to 70%) the total on-chip memory required.

Categories and Subject Descriptors

C.3 [**Special-purpose and application-based systems**]: Real-time and embedded systems; C.4 [**Performance of systems**]: Design studies and modeling techniques; I.6 [**Simulation and Modeling**]: Applications

General Terms

Performance, Design

Keywords

Video decoding, System-level design, Playout delay, On-chip memory

1. INTRODUCTION

Many system-on-chip platform architectures targeted towards the multimedia domain consist of multiple processing elements (PEs) connected by FIFO buffers in a pipelined fashion (e.g. Eclipse and Viper from Philips [1]). Each such PE executes a part of an application (e.g. Variable length decoding in an MPEG-2 decoder application) and runs concurrently with the other PEs. An important design constraint in such set-ups is to ensure that none of the FIFO buffers overflow and in addition, the playout buffer never underflows. To ensure the overflow constraint, PE stalls when the buffer it is writing to fills up. To prevent playout buffer underflow, the PE that writes to the buffer is usually clocked at a slightly higher frequency than the clock of the output device that reads the playout buffer.

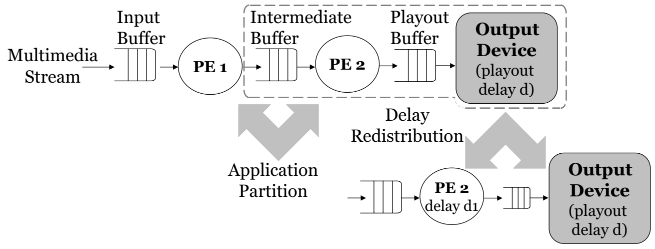


Figure 1: Our system model and technique. FIFO buffers connect PEs in pipeline. An application is partitioned and mapped onto the different PEs that run tasks concurrently. Buffer size reduces on redistributing playout delay.

Although a combination of these two techniques ensure both the buffer overflow and underflow constraints, its application involves additional stalling circuitry and the use of at least two different clock domains. To avoid these overheads, a common design tactic is to use a sufficiently large *playout delay* (i.e. the delay after which the output device starts reading the playout buffer) to avoid possible playout buffer underflows, in combination with large buffers to avoid buffer overflows. However, given the high variability in the execution requirements of most multimedia applications, the amount of buffer space required can be significant if one aims at a design with a buffer targeting the worst-case application.

In this paper, we propose a technique where the playout delay is not associated solely with the output device. Rather, this delay is redistributed over the multiple PEs in the pipeline. In other words, each PE starts reading its input buffer after a certain amount of time, or *delay*, has elapsed since the previous downstream PE started reading its input buffer. We show that such delay redistribution can significantly reduce the total buffer requirements, without increasing the total playout delay of the application being executed. Given that on-chip buffers occupy a large fraction of the chip area, our technique is useful in reducing chip area. At the same time it involves no extra implementation overheads.

Our delay redistribution scheme takes into account the variability in the execution requirements of an application and also the burstiness in the on-chip traffic. The manner in which the delay is distributed also depends on the partitioning of the application onto the multiple PEs. As a rule of the thumb, we propose that the total playout delay be redistributed among the different PEs in the architecture based on the ratio of the variability of the tasks running on the respective PEs. In other words, a PE running a task with high variability in its execution requirement should be associated with a higher delay. Although this basic technique is fairly intuitive, most current designs associate all the playout delay solely with the output device.

Our contribution in this paper is in conceiving the delay redistribution technique for multimedia pipelines and in formulating a mathematical framework that can guide a system designer in applying this scheme. We also present experimental results (based on an MPEG-2 decoder) to quantitatively show the reductions in the buffer size required after applying our technique. The buffer sizes we estimate are validated with the results obtained from simulation-based techniques.

Relation to previous work: Previous efforts [2, 10, 4, 5, 7, 3, 8] have specifically been directed towards optimizing on-chip memory in system-on-chip architectures designed for embedded multimedia systems. Most of the previous papers attempted to reduce memory requirements of synchronous data-flow (SDF) graphs which are used for specifying compute-intensive kernels of DSP applications. Murthy and Bhattacharya [5] proposed buffer merging to reduce memory requirements of SDF graphs. Buffer merging is achieved through sharing buffers that two different processes use. After analyzing the lifetime of actors (nodes specifying application code blocks in SDFs), it is determined whether two different processes containing these actors can potentially share buffers. Stuijk et al. [8] computed the pareto-optimal points that give the minimum storage space needed to execute a graph under given throughput constraints. We take a completely new approach in that we study the on-chip traffic characteristics of the application and exploit the playout delay parameter associated with the multimedia application to reduce the buffer size. Similar approaches have been followed in the domain of computer networks to counter the burst in network traffic so as to effectively utilize network resources. In comparison, our work is concerned with fixed playout delay, rather than dynamically adjusting it at runtime (as Ramjee et al. [6] studied). Further, our technique is more relevant in the context of playing stored audio and video. Hence, we did not exploit network related parameters such as loss and delay.

Organization of this paper: In the next section, using three illustrative scenarios, we show that the maximum buffer size required for the intermediate and the playout buffer reduces when the playout delay is redistributed among the display device and the PEs. To estimate the playout delay associated with the multimedia application, we propose a mathematical framework in Section 3. Using this system model, in Section 4, we compute the maximum initial playout delay required for a class of multimedia streams (e.g., same bit-rate/resolution video clips) such that the playout buffer never underflows. Then the playout delay is redistributed to the PEs, and the maximum total on-chip buffer size required before and after delay redistribution is estimated. To evaluate our analytical framework, we use an MPEG-2 decoder application as the case study in Section 5 and empirically show that the delay redistribution reduces the total on-chip memory required. Finally, we conclude the paper in Section 6.

2. ILLUSTRATIVE EXAMPLE

In this section, we first explain the mapping of the application to the multiprocessor system-on-chip architecture platform. Then, for this set-up (shown in Figure 1), the advantages of the delay redistribution technique over others is illustrated with some scenarios.

MPEG-2 decoder is the multimedia application we chose to map

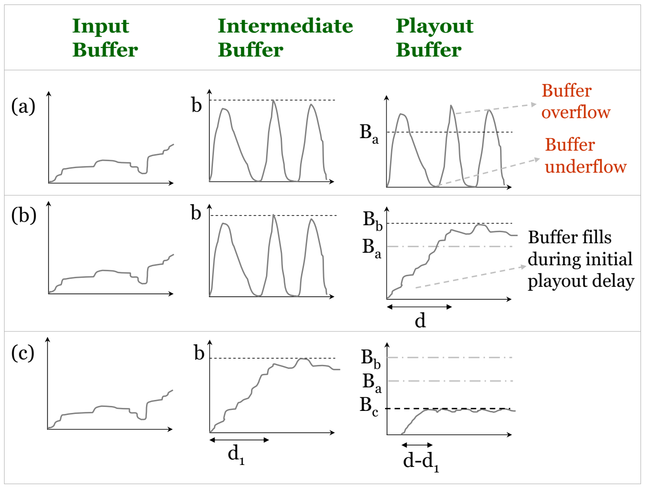


Figure 2: Buffer fill levels with initial playout delay: (a) very small, (b) large, and (c) redistributed.

to the system-on-chip platform. The sub-tasks of the application, namely, variable length decoding (VLD), inverse quantization (IQ), inverse discrete cosine transform (IDCT), and motion compensation (MC) are bound to the two on-chip processing elements (PE 1 and PE 2 in Figure 1). PE 1, running VLD and IQ, reads the input buffer, where the *stream objects* (or items) of the encoded input video arrive. PE 2, executing IDCT and MC, reads the partially processed stream objects (output of PE 1) from the intermediate buffer. The output device, after an initial delay d, reads the decoded stream objects (that PE 2 stored) from the playout buffer at a constant rate and displays the video. To show how our technique reduces the required on-chip memory size, we present three different scenarios. Figure 2 sketches the fill levels of the on-chip FIFO buffers in those scenarios. Scenario (c) is where we apply our delay redistribution technique, and scenarios (a) and (b) are existing techniques. Let us now discuss them in further detail.

Scenario (a), where the playout buffer underflows, occurs when the output device reads the playout buffer with no or small initial playout delay. The execution of some tasks of multimedia applications (e.g. VLD in MPEG-2) show high data-dependent variability. The PEs running these variable tasks (for e.g., PE 1 which is running VLD in our set-up) writes to their output buffer at a variable rate. In the example discussed here, the output buffer of PE 1 is the intermediate buffer and its fill level fluctuates through time (shown in Figure 2). This phenomenon of buffer fill level oscillation propagates to the playout buffer resulting in a *cascading effect*: varying fill levels at one buffer cause the fill level to vary at the next buffer in the pipeline, and this effect continues until it reaches the playout buffer. The output device, however, reads items from the playout buffer at a constant rate. Hence, it is possible that the playout buffer underflows resulting in a loss in quality of the displayed video. Also, due to the cascading effect, the intermediate buffer and the playout buffer might overflow if the buffers are not large enough. To avoid such overflows and underflows – as mentioned in Section 1 – the PEs have to stall (avoids overflows) and run at a higher clock frequency domain (avoids underflow at playout buffer). This technique has large overheads (e.g. in terms of stalling circuitry). In the next scenario, we will see how the initial playout delay absorbs the variabilities arising due to task execution (and due to variable event arrivals).

The display device starts reading from the playout buffer after a considerable initial playout delay in Scenario (b), and the playout buffer does not underflow. Only after the initial delay the output device starts consuming items from the playout buffer. Hence, during the delay, stream objects accumulate in the playout buffer. If this initial delay is appropriately chosen, then the variabilties occurring at the output of PE 1 will not propagate to the playout buffer (see playout buffer fill level in Figure 2 (b)). Since there are no variabilities in the fill level of the playout buffer, it never underflows. Effectively, we evaded the cascading effect. However, bursts at the intermediate buffer remains (see intermediate buffer fill levels in Figure 2). Now, like the playout buffer, what if the intermediate buffer plays the role of an *consumer*? That is, if we start PE 2 after a certain delay, then the variabilities at the output of PE 1 shall be absorbed at the intermediate buffer itself. The benefits of this delay redistribution are discussed next.

In Scenario (c), PE 1 initially starts to decode items. Then after a delay of d_1, PE 2 starts. Finally, after a delay d, the display device starts reading stream objects from the playout buffer. In Figure 2(c), we see no fluctuations in the intermediate buffer and the playout buffer, and the fill level at the playout buffer substantially reduces compared to Scenario (b). Note that the delay after which PE 2 should start (d_1) has to be appropriately chosen such that the intermediate buffer size does not increase after the delay redistribution. This is because the total buffer size required will not reduce in that case. For slight increase in the intermediate buffer size, our results show that the total maximum buffer size required reduces after redistribution. Since, at the intermediate buffer, unlike the playout buffer, the stream objects are partially compressed, they require less memory.

Problem statement: We solve the following problem using the analytical framework presented in the next section. Consider our system model shown in Figure 3, and assume a streaming application that is partitioned into tasks and mapped to the PEs of the system. A constant bit rate input stream arrives at the input buffer, PE 1 partially processes it, and writes it to the intermediate buffer. PE 2 reads items from the intermediate buffer, completely processes the stream, and writes it to the playout buffer. Finally, the display device consumes items from the playout buffer at a constant rate. Given the frequency at which PE 2 runs (f_2), the minimum frequency at which PE 1 should run (f_1), the corresponding minimum playout delay (d) is estimated such that the playout buffer never underflows. If the output device initially consumes items after delay (d), the playout buffer is guaranteed to never underflow. Then, preserving the playout buffer underflow guarantee, the maximum playout buffer size and intermediate buffer size required after delay redistribution should be estimated. It has to be shown that the total maximum buffer size required (sum of the intermediate buffer and playout buffer sizes) reduces after delay redistribution.

3. SYSTEM MODEL

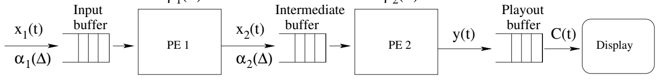


Figure 3: System Model

In this section, we model the variabilities in the arrival of input items and the variabilities in the processing requirement of the items.

We assume that the input bit stream to be decoded is fed into the input buffer at a constant rate of r bps. Further, for simplicity, we assume a stream consists of a sequence of *stream objects*. A stream object might be a macroblock in the case of video decoding. Now, given a media clip to be decoded, let $x_1(t)$ denote the number of stream objects arriving in the input buffer over the time interval $[0, t]$ (see Figure 3). Due to the variability in the number of bits constituting a stream object, the function $x_1(t)$ varies with the media clip. We define two functions $\alpha_1^l(\Delta)$ and $\alpha_1^u(\Delta)$ to bound the variability in the arrival process of the stream objects into the input buffer of PE 1 (see Figure 3). These two functions are defined as

$$\alpha_1^l(\Delta) \le x_1(t+\Delta) - x_1(t) \le \alpha_1^u(\Delta) \qquad (1)$$

for all t and $\Delta \ge 0$, where $\alpha_1^l(\Delta)$ and $\alpha_1^u(\Delta)$ denotes the minimum and maximum number of stream objects that can arrive at the input buffer within *any* time interval of length Δ, respectively.

To compute $\alpha_1^l(\Delta)$ and $\alpha_1^u(\Delta)$, we introduce two functions $\phi^l(k)$ and $\phi^u(k)$. The former denotes the minimum number of bits constituting *any* k consecutive stream objects in a bit-stream, and the latter denotes the corresponding maximum number of bits. These two functions can be obtained by analyzing a number of media clips that are *representative* of the clips to be processed by the target decoder.

Given the functions $\phi^l(k)$ and $\phi^u(k)$, it is possible to compute the *pseudo-inverse* of these two functions, denoted by $\phi^{l^{-1}}(n)$ and $\phi^{u^{-1}}(n)$, where the argument n is the number of bits. The functions $\phi^{l^{-1}}(n)$ and $\phi^{u^{-1}}(n)$ returns the maximum and minimum number of stream objects that can be constituted by n bits respectively. Since we assume the input bit stream arrives in the input buffer at a constant rate of r bps, we have

$$\alpha_1^l(\Delta) = \phi^{u^{-1}}(r\Delta) \text{ and } \alpha_1^u(\Delta) = \phi^{l^{-1}}(r\Delta).$$

We denote the arrival stream to the intermediate buffer as $x_2(t)$ and represent the alpha curves that bounds the variability in the arrival to PE 2 as $(\alpha_2^u(\Delta), \alpha_2^l(\Delta))$.

Similarly, we can characterize the variability in the number of processor cycles required to process any stream object using two functions $\gamma^l(k)$ and $\gamma^u(k)$. Both these functions, known as gamma curves, take the number of stream objects k as an argument. The function $\gamma^l(k)$ returns the minimum number of processor cycles required to process *any* k consecutive stream objects, and $\gamma^u(k)$ returns the corresponding maximum number of processor cycles. The gamma curves for PE 1 and PE 2 will be denoted as $\gamma_1^{l/u}$ and $\gamma_2^{l/u}$ respectively.

4. PLAYOUT DELAY REDISTRIBUTION

In this section, first we present our analytical framework to estimate the minimum playout delay required such that PE 1 and PE 2 run at minimum processor frequency required to meet the display rate. Later, we estimate the total maximum buffer size required (sum of the intermediate and the playout buffer sizes).

We assume that the playout buffer is readout by the output device at a constant rate of c stream objects/sec, after a playout delay (or buffering time) of d seconds. Let the function $C(t, d)$ be the number of stream objects readout by the output device over the time interval $[0, t]$ after playout delay d, then,

$$C(t, d) = \begin{cases} 0 & \text{if } t \le d \\ c(t-d) & \text{if } t > d. \end{cases} \qquad (2)$$

Now, given the input bitrate r, the functions $(\phi^{l/u}(k), \gamma_{1/2}^{l/u}(k))$ characterizing the possible set of media clips to be decoded, and the function $C(t, d)$, we can compute the minimum processor frequencies f_1 and f_2 to sustain the playout rate of c stream objects/sec. This is equivalent to requiring that the playout buffer never underflows.

Let $y(t)$ denotes the total number of stream objects written into the playout buffer over the time interval $[0, t]$. Then the playout buffer underflow constraint is equivalent to requiring that $y(t) \geq C(t, d)$ for all $t \geq 0$.

Let the *service* provided by PE 2 at frequency f_2 be represented by the function $\beta_2(\Delta)$. Similar to $\alpha_2^l(\Delta)$, $\beta_2(\Delta)$ represents the minimum number of stream objects that are guaranteed to be processed (if available in the intermediate buffer) within any time interval of length Δ. It can be shown that $y(t) \geq (\alpha_2^l \otimes \beta_2)(t), \forall t \geq 0$, where $\otimes$ is the *min-plus convolution* operator[1]. Hence, for the constraint $y(t) \geq C(t, d), \forall t \geq 0$ to hold, it is sufficient that the following inequality holds

$$(\alpha_2^l \otimes \beta_2)(t) \geq C(t, d), \ \forall t \geq 0. \tag{3}$$

It is known from the duality between $\otimes$ and $\oslash$, that for any three functions f, g, and h, $h \geq f \oslash g$ if and only if $g \otimes h \geq f$, where $\oslash$ is the *min-plus deconvolution* operator[2]. By applying this result on inequality (3), we obtain

$$\beta_2(t) \geq (C \oslash \alpha_2^l)(t, d), \ \forall t \geq 0. \tag{4}$$

Note that $\beta_2(t)$ in Inequality (4) is defined in terms of the number of stream objects that need to be processed within any time interval of length t. To obtain the equivalent service in terms of processor cycles, we can use the function $\gamma_2^u(k)$ defined above. The minimum service that needs to be guaranteed by PE 2 to ensure that the playout buffer never underflows is given by

$$\gamma_2^u(\beta_2(t)) = \gamma_2^u((C \oslash \alpha_2^l)(t, d)) \tag{5}$$

processor cycles for all $t \geq 0$. Hence, the minimum frequency at which PE 2 should run to sustain the specified playout rate is given by

$$\min\{f_2 \mid f_2 t \geq \gamma^u(\beta_2)(t)\}, \ \forall t \geq 0. \tag{6}$$

From the above equation, if PE 2 is run at frequency f_2, with a playout delay d (see Equation (2)), then it is guaranteed that the playout buffer will never underflow. Now, let us compute the minimum processor frequency for PE 1.

Considering that the playout delay is redistributed, the service that PE 2 should guarantee to meet the display requirement is given as

$$\beta_2(t) \geq \begin{cases} 0 & \text{if } t \leq d_1 \\ (C \oslash \alpha_2^l)(t, d) & \text{if } t > d_1, \end{cases} \tag{7}$$

with $d > d_1$.

PE 2 is idle during the initial delay d_1. After the initial delay, PE 2 produces items at a speed to sustain the display rate. To estimate the minimum playout delay required d, we will set d_1 to 0.

In the above equation, $\alpha_2^l(t)$ captures the minimum number of items that arrives to the buffer in-front of PE 2. So, $\alpha_2^l(t)$ lower bounds the number of items that PE 1 should produce and it can be written as

$$(\alpha_1^l \otimes \beta_1)(t) \geq \alpha_2^l(t), \ \forall t \geq 0. \tag{8}$$

Again, applying duality to the above equation we have

$$\beta_1(t) \geq (\alpha_2^l \oslash \alpha_1^l)(t), \ \forall t \geq 0. \tag{9}$$

[1]The min-plus convolution operator $\otimes$ is defined as follows. For any two functions f and g, $(f \otimes g)(t) = \inf_{0 \leq s \leq t}\{f(t-s) + g(s)\}$
[2]The min-plus deconvolution operator $\oslash$ is defined as follows. For any two functions f and g, $(f \oslash g)(t) = \sup_{s \geq 0}\{f(t+s) - g(s)\}$

Similar to Equation (6), let f_1 be the minimum processor frequency at which PE 1 should be run to guarantee an output of at least $\alpha_2^l(\Delta)$. Then, f_1 can be computed from the following equation as

$$\min\{f_1 \mid f_1 t \geq \gamma_1^u(\beta_1)(t)\}, \ \forall t \geq 0. \tag{10}$$

The frequency of PE 1 and PE 2, f_1 and f_2 respectively, depends on the playout delay d, which is a parameter of the consumption function (Equation (4)). So, we have to estimate the playout delay required to run the PEs at a minimum processor frequency computed in Equations (6) and (10). We denote $f(d)$ as the minimum frequency of the processing element corresponding to the playout delay d.

As shown in Figure 4, we define three different playout delays corresponding to processor frequency f_1 and f_2: initial, stabilization and maximum playout delays. The initial playout delay, d_i, is the largest playout delay below which the minimum processor frequency (for any PE in the pipeline) required to decode the stream is infinity. Maximum playout delay, d_m, is the smallest playout delay value above which there is no decrease in the minimum processor frequency (for all PEs in the pipeline) required to decode the stream and can be written as

$$\min\{d_m \mid f_i(d_m + \delta) = f_i(d_m)\}, \ \forall \delta \geq 0, \ \forall i. \tag{11}$$

The stabilization delay, which lies between the initial and the maximum delay, is the minimum playout delay required to run PE 1 and PE 2 at frequencies f_1 and f_2 and is defined as follows

DEFINITION 1. Stabilization playout delay (d_s) *is the delay value at which the minimum processor frequency (of all PEs in the pipeline) stabilizes to a value close to* $f(d_m)$. *It is defined for a given* ϵ *as follows:*

$$\min\{d_s \mid f_i(d_s) - f_i(d_m) \leq \epsilon\}, \ \forall i \geq 0. \tag{12}$$

In other words, PE 1 and PE 2 could run at a minimum processor frequency f_1 and f_2 respectively. But to sustain the playout rate, the display device should start consuming items from the playout buffer after the stabilization playout delay. Hence from the above equation, given the minimum processor frequencies f_1 and f_2, the minimum playout delay required can be estimated.

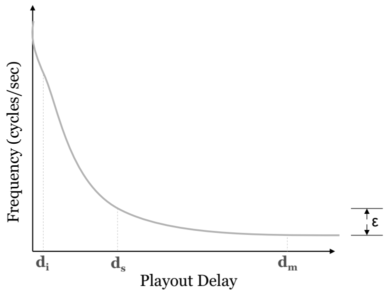


Figure 4: Initial playout delay values as minimum required processor frequency drops and stabilizes.

4.1 Buffer Size Estimation

To show that the buffer size reduces after redistributing the delay, we first have to compute the maximum intermediate and playout buffer sizes required. The following constraints must be satisfied when estimating the buffer sizes: (i) playout buffer should never underflow, and (ii) intermediate and playout buffer should never overflow.

To ensure that playout buffer never underflows, as we showed in the previous section, the processing elements have to run at the minimum processor frequencies f_1 and f_2 (From Equations (6 and 10)). The output device must start consuming items from the playout buffer after the stabilization delay d. Now, let us compute the maximum playout buffer and the intermediate buffer size required such that these buffers never overflow.

The playout buffer stores decoded stream objects that are to be consumed by the display device. We know that in time interval $[0, t]$, the number of stream objects processed is $y(t)$, and the number of stream objects consumed is $C(t, d)$. Hence the playout buffer size required is given by

$$B(d) = \max \{y(t) - C(t, d)\}, \forall t \geq 0. \quad (13)$$

We know that $y(t) \leq (\alpha_2^u \otimes \beta_2^u)(t)$ and hence the maximum playout buffer size required such that the buffer never overflows is given by

$$B(d) = \sup \{((\alpha_2^u \otimes \beta_2^u)(t) - C(t, d)\}, \forall t \geq 0. \quad (14)$$

In the above equation, the arrival of items to the playout buffer is variable, but the consumption rate is constant. The delay d, after which the playout device starts to consume items from the buffer, smoothens the variability in the fill levels of the playout buffer.

Similarly, the maximum intermediate buffer required is computed as follows

$$b_1(d, d_1) = \sup \left\{ (\alpha_1^u(t) \otimes \gamma_1^{l^{-1}}(ft)) - \beta_2^l(t) \right\}, \forall t \geq 0. \quad (15)$$

From the above equation, we see that both the arrival and the consumption of items from the intermediate buffer is at a variable rate. The variability in the fill level of the intermediate buffer due to the arrival of items to the buffer can be smoothened if PE 2 starts consuming items after appropriate delay d_1. This delay d_1 is redistributed from the playout delay d and in the next section we numerically show that the total maximum buffer size required reduces because of this redistribution.

5. EMPIRICAL EVALUATION

In this section, we present the results of our numerical analysis to show that the delay redistribution reduces maximum buffer size required. Then we present the simulation results that validates our model results. Now we explain the procedure for the experiments. We took MPEG-2 decoder as the multimedia application running in our multiprocessor system-on-chip (shown in Figure 3). There are two PEs - PE 1 partially decodes the incoming compressed video stream, and PE 2 fully decodes it and stores the decoded stream objects in the playout buffer. The decoding tasks that the two PEs run are sub-tasks of the decoder application. PE 1 runs VLD and IQ, and PE 2 runs IDCT and MC. This task mapping coincides with the software pipeline architecture of MPEG-2. For a given class of clips, using our analytical framework, we compute the play-out delay (d) and the delay after which PE 2 should start executing the tasks mapped to it. After this delay redistribution we show that the total buffer size (sum of the intermediate and the playout buffer) reduces. To estimate the playout delay, our framework needs input bit rate r, consumption rate c of the playout device, and functions ϕ and γ, which characterize the stream arrival and service requirements respectively. These functions are then used to obtain the playout delay and the amount of distribution in the playout delay required. Now, we first describe our simulation set-up that is used to obtain ϕ and γ for a class of video streams.

5.1 Experimental Set-up

We modeled our processor using the *sim-safe* configuration of the SimpleScalar instruction set simulator. MPEG-2 decoder source code was annotated with *start* and *stop* counters to record the number of processor cycles consumed by each stream object. To characterize the execution requirement of the decoder, we used a set of video clips having an average bitrate of 6000 kbps and a resolution of 704×480 pixels. The display rate of these clips was 30 fps. Figure 6 shows the three functions ϕ, α, and γ for this class of video clips. The procedure followed to obtain these functions is shown in Figure 5. Recall from Section 3 that ϕ characterizes the variability in the number of bits constituting each macroblock in the compressed video stream, α characterizes the variability in the arrival pattern of the video stream at the input buffer, and γ captures the variability in the execution requirement of each macroblock. Clearly, such a characterization is more expressive than traditional best/worst bounds, which are overly optimistic/pessimistic.

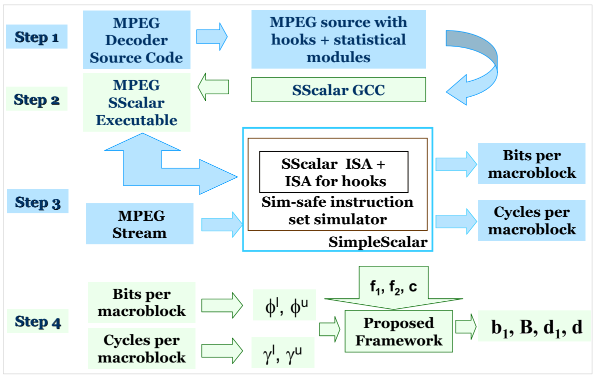


Figure 5: Experimental Set-up

5.2 Results and Validation

Frequency versus playout delay: We first compute the total initial playout delay required such that the processor frequency required for running tasks in PE 2 and PE 1 reduces to minimum. To find the delay, we first have to plug in Equation 10 the processor frequency that video decoding tasks in PE 2 need. Now using Equation 10, we compute the total playout delay required. Figure 8 shows the processor frequency f_1 versus delay for tasks running in PE 1 for two different values of f_2. We can now estimate the total playout delay to be the maximum delay such that processor frequency for tasks running in PE 1 reduces to minimum. So, if the output device starts after this initial playout delay the processor frequencies of tasks running in PE 1 and PE 2 drops to the lowest value possible. In Figure 8, for $f_2 = 460$ MHz, f_1 reduces from 100MHz to 40MHz as the total delay value increase from 100ms to 500ms. Hence d is 500ms. In our experiments, we chose numerous processor frequency values for PE 2 (i.e., for f_2). We show the results when $f_2 = 460, 500$ MHz because to execute the tasks mapped to PE 2, the average frequency at which PE 2 should run is 460 MHz. From Figure 8 it is evident even if f_2 is increased beyond the average frequency required for PE 2, the minimum required processor frequency f_1 is still 40 MHz. Hence there is no advantage in running PE 2 at a higher frequency than its average. On the other hand reducing f_2 below 460 MHz, we observed that (results not shown) the minimum required processor frequency for f_1 computed from Equation 10 is a large value than 40 MHz.

Buffer size reduction: Having found the total playout delay corresponding to minimum processor frequencies required for the processing elements, we now present results that shows memory reduction on delay redistribution. For the total playout delay d, the maximum playout buffer and intermediate buffer size required is estimated (Equation 14 and 15). The buffer sizes are estimated with and without redistributing the playout delay. The estimated max-

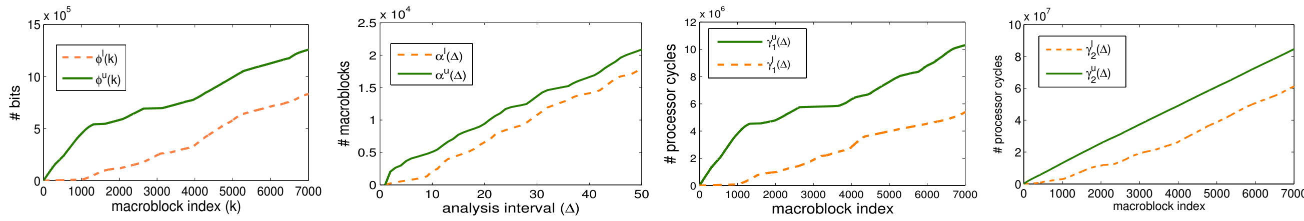
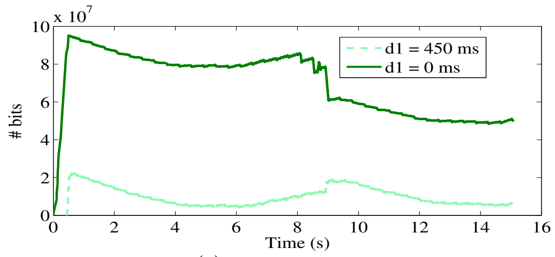
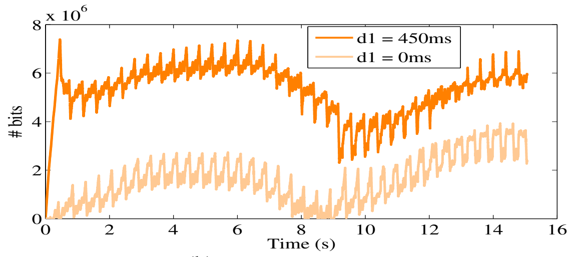


Figure 6: The functions ϕ, α and γ characterizing a high-bitrate, high-resolution class of video clips.



(a) Playout Buffer



(b) Intermediate Buffer

Figure 7: Change in buffer fill levels with redistributing playout delay.

imum buffer sizes from our analytical framework closely matches with that of the simulation results. Due to space constraints we are not presenting results that shows the accuracy of our model. We present a SystemC-based simulation results and show how much savings in buffer space we could actually obtain after delay redistribution. We used the video clip "cact" (obtained from [9]) in this simulation. The main result of this paper is shown in Figure 7(a) and Figure 7(b). The total buffer size savings (including the intermediate buffer) we obtained after delay redistribution for this clip is 70%. In Figure 7(a), the simulation results for $d_1 = 0ms$ and $d_1 = 450$ms is shown. Recall that d_1 is the delay after which PE 2 starts processing. As we could see in this figure, the playout buffer fill level substantially reduces after redistributing the delay. Figure 7(b) shows the fill level of the intermediate buffer over time, and we see that after the redistributing delay, the fill level of the intermediate buffer increases. Note here that irrespective of the increase in the fill level of the intermediate buffer, the total buffer size in terms of bits does not increase. In fact, it reduces. Since the partitioned decoded macroblocks in the intermediate buffer occupy less memory as compared to the playout buffer.

Note on decoder source code instrumentation: We instrumented the libmpeg2 decoder source code to obtain the bits corresponding to partially decoded macroblock. After VLD and IQ, the DCT blocks corresponding to each macroblock will have several zero and non-zero coefficients. Instead of storing each block in matrix format at the intermediate buffer, we only store the location and the value of each coefficient. We needed maximum 16 bits to store a coefficient and 6 bits to store the location of the coefficient, leading to a more efficient storage.

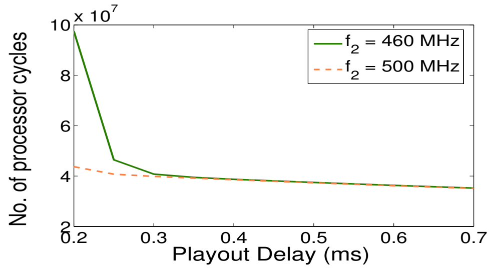


Figure 8: Playout delay estimation w.r.t processing requirement of tasks (VLD and IQ) running in PE 1.

6. CONCLUSIONS

A novel technique to reduce the on-chip memory size required for stream processing on multiprocessor system-on-chip architectures is proposed in this paper. Playout delay associated with the display device in a multimedia embedded system is redistributed to the processing elements on-chip connected in pipeline to the output device. This delay redistribution reduced the maximum on-chip memory required because the variabilities in the buffer fill levels (due to event arrival and task execution at the PEs) were stopped from propagating to subsequent buffers. We presented a mathematical framework, using which we could estimate the total playout delay required and the delay to be redistributed. We validated our results using simulation, and we obtained up to 70% savings in buffer size after applying our technique.

7. REFERENCES

[1] M. J. Rutten et al. Eclipse: Heterogeneous multiprocessor architecture for flexible media processing. In *IPDPS*, Fort Lauderdale, FL, April 2002.

[2] P. R. Panda et al. Data and memory optimization techniques for embedded systems. In *TODAES*, volume 6, pages 149–206, New York, NY, 2001. ACM Press.

[3] S. Han, X. Guerin, S.Chae, and A. A. Jerraya. Buffer memory optimization for video codec application modeled in simulink. In *DAC*, San Francisco, CA, July 2006.

[4] D. Ko and S. S. Bhattacharyya. Modeling and optimization of buffering trade-offs for hardware implementation of image processing applications. In *IEEE Workshop on Signal Processing Systems Design and Implementation*, Athens, Greece, November 2005.

[5] P. K. Murthy and S. S. Bhattacharyya. Buffer merging- A powerful technique for reducing memory requirements of synchronous dataflow specifications. In *TODAES*, volume 9, pages 212–237, New York, NY, 2004. ACM Press.

[6] R. Ramjee, J. Kurose, D. Towsley, and H. Schulzrinne. Adaptive playout mechanism for packetized audio applications in wide area networks. In *INFOCOM*, Toronto, Canada, June 1998.

[7] N. Sarshar and X. Wu. Buffer size reduction through buffer sharing for streaming applications. In *ICME*, Taipei, Taiwan, June 2004.

[8] S. Stuijk, M. Geilen, and T. Basten. Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs. In *DAC*, San Francisco, CA, July 2006.

[9] Tektronix. ftp://ftp.tek.com/tv/test/streams/Element/index.html.

[10] H. Yang, H. Jung, and S. Ha. Buffer minimization in RTL synthesis from coarse-grained dataflow specification. In *SASMI*, Nagoya, Japan, April 2006.