Identifying "Representative" Workloads in Designing MpSoC Platforms for Media Processing

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

Abstract— **Workload design is a well recognized problem in the domain of microprocessor design. Different program characteristics that influence the selection of a representative workload include microarchitecture-centric properties such as cache miss rates, instruction mix and accuracy of branch prediction. However, properties of a workload that are pertinent to the context of system-level design of multiprocessor SoC platforms are very different. Till date, the problem of "representative workload design" in this specific context has not been sufficiently addressed. This paper represents an attempt to address this problem in the specific case of SoC platform design for multimedia processing. Towards this, we present a method to characterize properties of multimedia workload that are relevant to SoC platform design. Based on such a characterization, we present a technique for classifying different multimedia streams. Finally, we show the utility of such a classification through a case study involving the design of a multiprocessor SoC platform for MPEG-2 decoding.**

I. Introduction

Recently there has been a lot of interest in designing multiprocessor System-on-Chip (MpSoC) platform architectures for multimedia processing. Examples of such architectures are Eclipse and Viper from Philips, which are targeted towards advanced set-top boxes and DTVs. A typical design process of such an architecture involves a thorough exploration of the available design space. During this design space exploration, many different platform configurations are evaluated and compared with each other with the goal of identifying an architecture that would be most suitable for a set of target applications. Usually this process heavily relies on system simulations as a means of performance estimation of the candidate architectures. Ideally, each implementation of an application on a MpSoC architecture has to be evaluated for a large number of possible inputs. However, this is an expensive process since the simulation involved for each input might require a considerable amount of time. For example, simulation of only a few minutes of video, for a video decoding application, may consume tens of hours [12]. This significantly limits the number of different inputs for which simulations can be performed within an allotted design time. Therefore, from a large library of possible inputs, the system designer has to choose the smallest subset which is "representative" of the workload that the architecture would experience in practice. Simulations can then be restricted to this subset only.

Selecting a good "representative" input set is of course not a new concern—benchmark selection or workload design is a well recognized problem in the domain of microprocessor design. However, the main issues in that domain are microarchitecture-centric, where a designer is mostly concerned with program characteristics like instruction mix, data and instruction cache miss rates and branch prediction accuracy. On the other hand, the concerns in the case of system-level design of platform architectures are very different and these are not suitably reflected in a benchmark suite designed for microarchitecture evaluation.

In this paper we attempt to address this issue of workload design in the specific context of system-level design of platform architectures for multimedia processing. Although simulation-oriented design and evaluation are widespread in the domain of system-level design, to the best of our knowledge the issue of methodically selecting representative inputs for architecture evaluation has not received any attention so far. Most of the work reported in the Embedded Systems literature, on novel system models or simulation schemes, shirk off this problem and leave the responsibility of choosing a representative input or stimuli to the architecture on the system designer (see, for example, [6]).

There are many reasons why this problem is interesting in the specific case of multimedia processing on MpSoC platforms. Firstly, many multimedia applications exhibit a large degree of data-dependent variability that complicates the problem of choosing a representative input set. Secondly, in contrast to general-purpose architectures, MpSoC platforms that are optimized for stream processing have heterogeneous parallel architectures. This fact further complicates the problem. Thirdly, multimedia processing is in general computationally intensive, which makes workload selection an important problem.

Arbitrarily selecting inputs to form the "representative" input set is certainly not a good idea. The goal of "representative" workload design should be to select inputs that represent *corner cases* for the target architecture, i.e. those inputs which impose worst- and best-case loads on different parts of the architecture. However, determining what constitutes a "corner case" is not a trivial undertaking due to the complex nature of most multimedia workloads. Attempts towards using some qualitative technique to judge the properties of multimedia streams based on their content (for example, by simply viewing video clips to be processed by the architecture and classifying them based on experience or intuition) might easily fail. Hence, a quantitative methodology is necessary, using which it should be possible to objectively assess and compare the properties of different multimedia streams. Based on such a comparison, a small *representative* subset of a large library of samples can then be chosen.

In this paper we propose such a methodology to classify multimedia streams, which can be used to identify a small representative set meant for architecture evaluation. Towards this, we first hypothesize that all the characteristics of multimedia streams that influence the performance of a MpSoC platform architecture, are related to their "variability". Such variability manifests itself

0-7803-8631-0/04/$20.00 ©2004 IEEE.

as data-dependent fluctuations of (i) execution time requirements and (ii) input-output rates associated with multimedia processing tasks. These fluctuations stem from the fact that execution time requirements of the tasks and the amount of data consumed and produced by the tasks depend on the properties of particular audio/video samples being processed. Now, given a library of multimedia streams, we classify two streams as *similar* if both of them exhibit the same kind of variability with respect to execution time requirements and input/output rates as mentioned above. Therefore, given a set of video streams which are *similar*, it would be sufficient to simulate an architecture with only one video stream from this set, as all the other streams would "stress" the architecture in the same way. To quantitatively characterize the variability associated with a stream, with respect to an architecture, we use a novel concept called *variability characterization curves* (VCCs) [9] which is summarized in Section II. As an illustration of our methodology, throughout the presentation we use a case study of an MPEG-2 decoder application.

We would like to point out here that the kinds of variabilities that should be considered in a multimedia stream for an effective classification would depend on the architecture and the application at hand, and a detailed discussion of this is beyond the scope of this paper. The contribution of this paper is to point out that the properties of multimedia streams, that should be considered for representative workload identification in the context of performance evaluation of SoC platforms, can be expressed in the form of VCCs.

Related Work: The construction of representative workloads for performance evaluation of computer systems has always been an area of active research since early 70s (see [11] and references therein). Since then the term *workload* has been widely understood as a mix of programs (or jobs, or applications) for which the performance of a computer system was evaluated. Domain-specific collections of such programs, called *benchmarks*, have been designed and widely used as a standard means to evaluate and compare computer architectures. Examples of these are MediaBench [8] and the Berkeley multimedia workload [10]. Design of such representative workloads was mainly concentrated on proper selection of the *programs* to be included in the workload. The selection of corresponding input data sets was limited to the definition of their size (e.g. sampling rate, resolution etc.) The dependency of program behavior on the values of the input data sets did not receive enough consideration in the process of forming such representative workloads.

Recently Eeckhout et al. [1] have shown that the *workload design space* may be very complex and therefore should be systematically explored during the construction of representative workloads. Their workload design space consists of *program-input pairs* that capture both, the variety of programs as well as various input data sets to those programs. They use techniques such as principle component analysis and cluster analysis to efficiently explore the space of possible workloads and select representative program-input pairs from it.

The problem of reducing simulation time has been addressed using *trace sampling techniques* (see [5] and references therein). The goal of such techniques is to identify representative fragments in the program execution and simulate only those fragments, thereby eliminating the need for simulating the entire program. Trace sampling techniques heavily rely on the characterization and classification of the workload imposed on the architecture by the different fragments in the program execution trace. However, it should be noted that all the above mentioned research efforts were primarily targeted towards characterization and composition of representative workloads in the domain of microprocessor design.

II. Workload Characterization

Clearly, workload characterization should be based on *key properties* that are important in a particular design context. Usually these are properties that have a strong impact on the performance of the architecture being designed. For instance, in microarchitectural design such properties would be instruction mix, branch prediction accuracy and cache miss rates [1]. As mentioned in the previous section, our hypothesis is that *on the system level* the performance of multimedia MpSoC architectures is largely influenced by various kinds of *data-dependent variability* associated with the processing of multimedia data streams. This hypothesis rests on the observation that such variability is the major source of the burstiness of on-chip traffic in such multimedia MpSoC platforms [12]. The burstiness of the on-chip traffic necessitates the insertion of additional buffers between architectural entities processing the multimedia streams, and the deployment of sophisticated scheduling policies across the platform. Both of these inevitably translate into increased design costs and power consumption [3]. Therefore, it is certainly meaningful to characterize multimedia workloads w.r.t. their variability properties.

In this section we present a generic model that allows us to quantitatively capture the variability found in multimedia streams. It is based on the concept of *variability characterization curves* (VCCs), details of which may be found in [9]; this concept is briefly explained at the end of this section. But first we briefly outline the structure of MpSoC platforms that we consider in this paper and pinpoint the sources of workload variability associated with the processing of multimedia streams on such platforms.

Platform architecture: A typical MpSoC platform consists of a heterogeneous collection of interconnected processing elements (PEs) such as programmable processors and coarse-grained application-specific co-processors. An application to be executed on the platform is split into concurrent tasks that are assigned for execution to different PEs of the architecture. The tasks communicate with each other via unidirectional data streams. Such data streams consist of a sequence of *stream objects*, i.e. units of data consumed from (or produced to) a communication channel by a task, when it is activated.

An example of such an MpSoC platform is shown in Figure 1. The platform consists of two programmable processors, PE_1 and PE_2, and input and output interfaces. Figure 1 also shows a mapping of a MPEG-2 decoder application on to the platform. PE_1 executes a task performing VLD and IQ functions, whereas PE_2 executes a task performing IDCT and MC functions of the MPEG-2 decoding algorithm. For the sake of brevity, we will refer to these tasks as the *VLD task* and the *IDCT task* respectively. In the system shown in Figure 1, stream objects belonging to the input stream emerging from the network interface are single bits. Stream objects sent from PE_1 to PE_2 are partially decoded

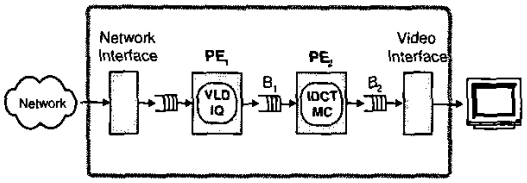

Fig. 1. MpSoC platform onto which an MPEG-2 decoder application is mapped

macroblocks, whereas stream objects entering the video interface are fully processed macroblocks.

What are the sources of variability that are usually associated with the processing of multimedia streams on such MpSoC platforms? Firstly, arrival patterns of multimedia streams at the input of the system may have a bursty nature, i.e. stream objects may arrive on the system's input in highly irregular intervals. A typical example of this is a multimedia device receiving streams from a congested network. Secondly, each activation of a task may consume and produce a variable number of stream objects from the associated streams. For example, each activation of the VLD task in Figure 1 consumes a variable number of bits from the network interface, although, it always produces one macroblock at its output. Thirdly, the execution demand of a task may vary from activation to activation due to data-dependent program flow. Both the tasks in our running example of the MPEG-2 decoder—VLD and IDCT—possesses this property. Finally, stream objects belonging to the same stream may require different amounts of memory to store them in the communication channels. Again, in the example architecture shown in Figure 1, we note that the partially decoded macroblocks stored in buffer B_1, depending on their type, may or may not include motion vectors.

All these types of variability must be carefully considered and characterized during the workload design process. In this paper, we will be concerned with the variability of the execution demand and the consumption and production rates of tasks. As mentioned before, depending on the architecture and the application at hand, it might be meaningful to consider other types of variabilities as well. However, we show that the two types we consider here already lead to meaningful results.

Variability characterization curves: The basic concept behind VCCs have been introduced and fully described in [9], where VCCs were used for analytical performance evaluation of MpSoC platforms. In this paper we present a fundamentally different application of this concept.

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 $\gamma = (\gamma^l, \gamma^u)$: Suppose we are given a stream which is to be processed by a task T implemented on a PE. The execution requirement 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 processor cycles of the PE which is required to complete a sequence of k executions of T. $\gamma^u(k)$ is the maximum number of processor cycles which may be necessary to complete a sequence of k executions of T.

Consumption and production curves $\kappa = (\kappa^l, \kappa^u)$ and $\pi = (\pi^l, \pi^u)$: Let an input stream be processed by a task T. 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.

On the other hand, 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. $\pi^u(k)$ 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.

III. Classification of Streams

We propose to classify streams based on the *shapes* of the VCCs associated with them. If two streams are characterized by VCCs having similar shapes, then their behaviors, in the worst/best-case, will also be similar. Each stream might be associated with several types of VCCs, characterizing different aspects of variability within the stream. Therefore, if two streams have similarly shaped VCCs of respective types, then they will impose similar workload on the architecture (in the worst- and best-case). For example, the maximum backlogs that such streams will create in the buffers of the architecture as a result of their processing will almost be the same.

For the classification of streams based only on a single variability type, we first define a measure of *dissimilarity* between two VCCs of the same type. In general, any measure of dissimilarity between two objects depends on the specific problem at hand. Each property, based on which two objects are to be compared, is associated with a variable. Any valuation of the variables associated with a set of properties, then constitutes a representation

of an object. The dissimilarity between two objects is found by computing some *metric* defined over these variables. In our case, a VCC, which is defined for a set of points $k = 1, 2, .., n$, can be seen as an object described by n variables. Intuitively, to see how dissimilar the shapes of two VCCs (of the same type) are, we need to compare their values for each of the points $k = 1, 2, .., n$. By noting that all n variables represent a VCC along essentially *separable* dimensions, we can quantitatively measure the dissimilarity between two VCCs using the City Block metric [2]. We decided to use this metric as a measure of dissimilarity because in comparison to other known metrics (e.g. Euclidean Distance) it is more "sensitive" to differences in each of the dimensions, i.e. in our case, the metric is more "sensetive" to the differences in the shapes of two VCCs. Given below is a formal definition of the dissimilarity between two VCCs, based on the City Block metric.

Let $\theta_{ri}(k)$ $(k = 1, 2, .., n)$ denote a VCC of type r associated with the ith stream. A measure of the pairwise dissimilarity between two streams i and j, with respect to a VCC of type r, is then defined as

$$d_{rij} = \sum_{k=1}^{n} \omega_r(k) \, |\theta_{ri}(k) - \theta_{rj}(k)| \qquad (1)$$

where $\omega_r(k) = 1/k$ are weights that are necessary to normalize the differences $|\theta_{ri}(k) - \theta_{rj}(k)|$ w.r.t. the length k of the analysis interval. The longer the analysis interval, the less *critical* is the difference in the values of the two VCCs corresponding to this interval. For example, suppose that we want to compare the upper workload curves of two streams. From the workload curves, suppose we know that any two consecutive stream objects (i.e. $k = 2$) from the first stream may cause a maximum execution demand of 100 units, while for the second stream this value is 150. Suppose that we also know that any 10 consecutive stream objects ($k = 10$) from the first stream may cause a maximum execution demand of 1000 units, and for the second stream it is 1150 units. Although the absolute difference between the curves for $k = 2$ is smaller than that for $k = 10$, the difference in the execution demand computed *per stream object* for $k = 2$ is larger than for $k = 10$ ($\frac{|100-150|}{2} > \frac{|1000-1150|}{10}$). For $k = 10$ the absolute difference is *distributed* over a larger number of stream objects than in the case of $k = 2$, and therefore this difference becomes less critical.

In many cases it might be useful to characterize streams using more than one type of VCCs. How should the dissimilarity between streams be quantified in such cases? We believe that first, the measure of dissimilarity between VCCs having identical types should be computed using Eqn. (1). These measures can then be combined in various possible ways, one of them being simply computing the sum of all the dissimilarity measures for the individual VCC types. The pairwise dissimilarity between two streams i and j w.r.t. VCCs of types $r = 1, 2, .., p$ is then defined as

$$d_{ij} = \sum_{r=1}^{p} d_{rij} \qquad (2)$$

To *classify* streams using the dissimilarity measure described above, we use a conventional hierarchical clustering algorithm based on the *complete linkage* algorithm [2] for computing distances between clusters. The rationale behind the choice of the complete linkage algorithm is the need to keep the clusters as dense as possible.

index	video clip	index	video clip
1	100b_080.m2v	7	pulb_080.m2v
2	bbc3_080.m2v	8	susi_080.m2v
3	cact_080.m2v	9	tens_080.m2v
4	flwr_080.m2v	10	time_080.m2v
5	mobl_080.m2v	11	v700_080.m2v
6	mulb_080.m2v		
Source: ftp.tek.com/tv/test/streams/Element/MPEG-Video/			

TABLE I
MPEG-2 VIDEO CLIPS USED IN OUR EXPERIMENTS

IV. Empirical Validation

To see how the stream classification method described in the previous section performs on real data samples, we conducted a number of experiments with MPEG-2 video streams. Since MPEG-2 streams have a complex nature and a rich set of characteristics [4], they represented an interesting target for our experiments.

The goal is to customize a generic MpSoC platform, such as the one shown in Figure 1. The platform has to be customized such that it supports real-time decoding of MPEG-2 video streams. Hence, we need to study the impact of different MPEG-2 streams on the platform and based on the results of our study, optimize the architecture accordingly. For this purpose we collected a large library of video clips that we believe our architecture should be able to support. However, due to time constraints we cannot afford to run simulations for all the clips in the library. Furthermore, simulation of an entire clip takes a prohibitively long time. Therefore, we are constrained to simulate only a limited number of *short fragments* extracted from *selected* video clips belonging to the library.

We assume that any video clip in the library contains only one *scene*. In a visual sense, a scene is "*a portion of the movie without sudden changes in view, but with some panning and zooming*" [4]. Distinguishing between different scenes is necessary, because even within a single MPEG-2 stream different scenes might have substantially different characteristics. For example, characteristics of MPEG-2 streams (such as bit rate) may *significantly* vary at a large time scale, i.e. across different scenes, while at a short time scale (i.e. within a scene) the variations are more moderate [4], [7]. If different scenes are not treated separately while deriving their VCCs, due to the nature of VCCs, important information about some scenes may be overshadowed by other scenes. Finally, we note that in practice it is always possible to split a long movie into a series of individual scenes (see [4] for the relevant references).

For our experiments, we used a library of MPEG-2 video clips that is shown in Table I. Each clip in the library is a 8 Mbps constant bit rate stream consisting of only one scene with a resolution of 704×576 pels and a frame rate of 25 fps. We believe that the variety of scenes represented by this library is sufficient for a demonstration of our classification method.

To select representative streams for performance evaluation of our architecture, we classified the streams in the library based on (i) the variability in execution demand, and (ii) the variability in the production and consumption rates of the tasks running on the PEs of the platform. The VLD task has both these types of variabilities. For each activation, it consumes a variable number of

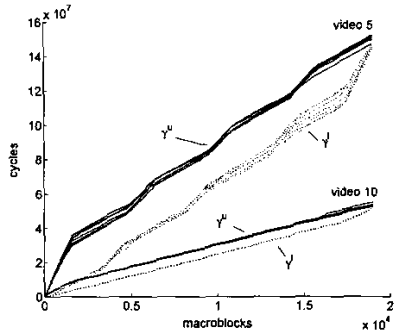

Fig. 2. $(\gamma^u_{vld}, \gamma^l_{vld})$ for different fragments of video 5 and video 10

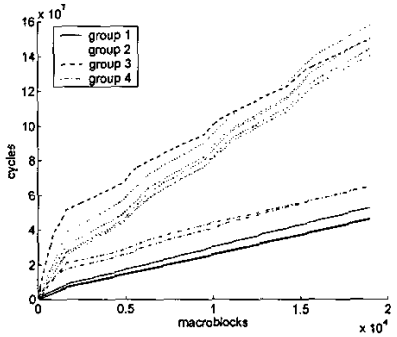

Fig. 3. Classification based on γ^u only

VCC	max.dissim	video	VCC	max.dissim	video
γ^u_{vld}	57151356	4	γ^l_{idtc}	37220944	3
γ^l_{vld}	23548299	4	κ^u_{vld}	2146073	4
γ^u_{idct}	22903156	9	κ^l_{vld}	752238	4

TABLE II
MAXIMUM DISSIMILARITY BETWEEN FRAGMENTS OF THE SAME SCENE

bits from the input buffer and its execution demand also fluctuates. Hence, we characterized it using the workload curves $(\gamma^u_{vld}, \gamma^l_{vld})$ and the consumption curves $(\kappa^u_{vld}, \kappa^l_{vld})$. The IDCT task was characterized using only the workload curves $(\gamma^u_{idct}, \gamma^l_{idct})$, because its execution demand is variable but consumption and production rates are constant.

Experimental Setup: Our simulation environment consisted of the SimpleScalar instruction set simulator, a system simulator and a MPEG-2 decoder program. The MPEG-2 decoder program was used as an executable for the simulator and as a means to obtain traces of bit allocation to macroblocks. The instruction set simulator was used to obtain traces of execution times for the VLD and IDCT tasks of the MPEG-2 decoding algorithm. Both tasks worked at the macroblock granularity. The system simulator consisted of a SystemC transaction-level model of the architecture shown in Figure 1. We used it to measure backlogs in the buffers resulting from the execution of the MPEG-2 decoder application on the platform.

We used the *sim-profile* configuration of the SimpleScalar simulator and the PISA instruction set to model PE_1 and PE_2 of the architecture. Although this configuration does not model advanced microarchitectural features of the processor, it allows fast simulation and was therefore the most suitable choice. This choice is also justified by the fact that advanced features in the microarchitecture of a general purpose processors do not have significant impact on the variability of multimedia workloads [3].

The VCCs were obtained from the collected execution traces. To obtain an upper (lower) VCC we searched through the corresponding trace with time windows of different lengths and identified the maximum (minimum) execution requirements (or number of bits) occurring in the trace within each of these time windows. The maximum window size was determined by the maximum time interval over which the streams were compared. For each design scenario, this might be different. In our experiments we had set the maximum window size to 12 frames. This corresponds to the most frequently occurring length of group of pictures (GOP) in the MPEG-2 bitstreams.

Note that obtaining the VCCs relies only on the instruction set simulation and a simple trace-analysis algorithm, both of which are orders of magnitude faster compared to a full system simulation. Our proposed method therefore results in considerable savings in design time.

Results and Discussion: Our first step was to compute the maximum dissimilarity between VCCs obtained from different fragments of the same scene. If this dissimilarity is sufficiently low then we can randomly pick a short fragment from a long video clip and use it as a representative of the whole video clip. If this dissimilarity is too high, then we may need to adopt other approaches to select short fragments. For example, fragments of the same scene can be classified first. Then *several* fragments can be chosen to represent that scene.

From each clip in our library, we extracted 10 unique fragments of the same length (30 frames) and measured their VCCs. Figure 2 shows results of the measurements for $(\gamma^l_{vld}, \gamma^u_{vld})$ for two video clips, i.e. clip numbers 5 and 10 from Table I. Video 5 represents a natural full-motion scene, whereas video 10 is a video test pattern displaying a small running timer on a still background. By inspecting the plots in Figure 2 we can see that the dissimilarity between fragments of video 5 is larger than those between fragments of video 10. This can be explained by the higher degree of motion present in the scene of video 5. Nevertheless, we can see that the curves for different fragments of video 5 exhibit a similar behavior. For other videos in the library, we observed very similar trends.

Using Eqn. (1) for each VCC type we computed pairwise dissimilarities between fragments of the same scene and selected their maximum value. Table II shows the obtained maximum values taken *over all* the video clips. From this table we can observe that video 4 probably contains a very complex and changing scene, because almost all the VCC types of its fragments exhibit a higher dissimilarity compared to those for the other clips.

For the classification of the (full length) video clips we decided to randomly pick one fragment from each clip and then perform

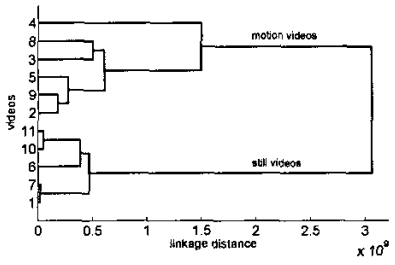

Fig. 4. Cluster tree

the classification based only on the selected fragments. For the purpose of illustration, we first performed the classification based on only *one* VCC type, γ^u. The results of the classification into four groups, based on the shape of γ^u, are presented in Figure 3. As we can see in the figure, our method could correctly identify groups of curves having similar shapes. This indicates that the measure of dissimilarity defined by Eqn. (1) leads to a meaningful classification.

Figure 4 shows a *dendrogram* of the hierarchical cluster tree obtained as a result of the classification based on *all* VCC types, i.e. by using Eqn. (2). In this dendrogram we can clearly distinguish between two major groups of clips: still and motion videos[1]. This kind of a coarse-grained division into two groups would have been possible to obtain just by viewing the videos on the screen. However, a more refined classification would be difficult to achieve using such a subjective technique. For example, before performing the experiments, by simply viewing the clips we could not predict that video 4 would have such different properties in comparison to the other motion videos. However, we can easily see this in the dendrogram: all other motion videos except video 4, form a tight cluster with the maximum linkage distance almost three times smaller than the maximum linkage distance when video 4 is included into the cluster.

Finally, to see how the results of the stream classification correlate with the actual impact of the streams on the architecture, we performed simulations of the system shown in Figure 1. We simulated the decoding of several *full-length* video clips from our library. As a measure of the *architectural impact* we decided to use maximum backlogs occurring as a result of the MPEG-2 processing in the buffers B_1 and B_2. The backlog in the buffer in front of PE_1 was not taken into account due to its relatively small size.

Table III summarizes the simulation results. Our measurements show that, for example, videos 1 and 7 produce very similar maximum backlogs in the both buffers. The maximum backlogs produced by videos 9 and 2 are *less similar* than the backlogs produced by videos 1 and 7. For videos 9 and 2, the differences in the backlogs in B_1 and B_2 are 2110 and 245 macroblocks respectively. We can also see that video 9 is *more similar* to video 2 than to video 3. The maximum backlogs for video 3 and video 9 differ by 4935 and 405 macroblocks in B_1 and B_2 respectively. Hence, we can see that the simulation results exhibit the same tendency as that shown by the classification in Figure 4.

[1]Since video 10 is mostly still, it was assigned to the group of still videos by our method.

video	B_1	B_2	video	B_1	B_2
1	8282	9433	4	4443	8732
2	5128	9027	7	8390	9593
3	7953	8867	9	3018	9272

TABLE III
MEASURED MAXIMUM BUFFER BACKLOGS

V. Concluding Remarks

In this paper we presented a promising approach for workload design for the specific context of system-level design of MpSoC platforms. Our two main contributions were: (i) identifying VCCs as a means for representing properties of multimedia workloads for system-level design of media processing platforms, and (ii) a classification method based on VCCs to cluster multimedia streams which exert similar influences on a platform architecture. We presented preliminary results that show the usefulness of this approach. However, there is considerable scope for further research in this direction. For example, a more systematic study needs to be done to identify "variability types" beyond the ones considered in this paper. We are not aware of any previous work in this direction and hope that this paper will encourage a systematic study of this problem, especially since simulation time is a widely recognized deterrent in the case of simulation-based performance evaluation of embedded systems.

Acknowledgements: This work is 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] L. Eeckhout, H. Vandierendonck, and K. De Bosschere. Workload design: Selecting representative program-input pairs. In *IEEE PACT*, pages 83–94, 2002.

[2] A. D. Gordon. *Classification*. Chapman & Hall/CRC, 1999.

[3] C.J. Hughes, P. Kaul, S.V. Adve, R. Jain, C. Park, and J. Srinivasan. Variability in the execution of multimedia applications and implications for architecture. In *ISCA*, pages 254–265, 2001.

[4] M. Krunz and S.K. Tripathi. On the characterization of VBR MPEG streams. *SIGMETRICS Perform. Eval. Rev.*, 25(1):192–202, 1997.

[5] T. Lafage and A. Seznec. Choosing representative slices of program execution for microarchitecture simulations: a preliminary application to the data stream. In *Workload characterization of emerging computer applications*, pages 145–163. Kluwer Academic Publishers, 2001.

[6] 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.

[7] A.A. Lazar, G. Pacifici, and D.E. Pendarakis. Modeling video sources for real-time scheduling. *Multimedia Syst.*, 1(6):253–266, 1994.

[8] C. Lee, M. Potkonjak, and W.H. Mangione-Smith. MediaBench: a tool for evaluating and synthesizing multimedia and communicatons systems. In *ACM/IEEE MICRO*, pages 330–335, 1997.

[9] A. Maxiaguine, Y. Zhu, S. Chakraborty, and W.-F. Wong. Tuning SoC platforms for multimedia processing: Identifying limits and tradeoffs. In *CODES+ISSS*, 2004. To appear.

[10] N.T. Slingerland and A.J. Smith. Design and characterization of the Berkeley multimedia workload. *Multimedia Syst.*, 8(4):315–327, 2002.

[11] K. Sreenivasan and A. J. Kleinman. On the construction of a representative synthetic workload. *Commun. ACM*, 17(3):127–133, 1974.

[12] G.V. Varatkar and R. Marculescu. On-chip traffic modeling and synthesis for MPEG-2 video applications. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 12(1):108–119, January 2004.