Automated Hardware-Independent Scenario Identification

Juan Hamers
jmhamers@elis.ugent.be

Lieven Eeckhout
leeckhou@elis.ugent.be

ELIS Department
Ghent University, Belgium

ABSTRACT

Scenario-based design exploits the time-varying execution behavior of applications by dynamically adapting the system on which they run. This is a particularly interesting design methodology for media applications with soft real-time constraints such as decoders: frames can be classified into scenarios based on their decode complexity, and the system can be configured on a per-scenario basis such that energy consumption is reduced while still meeting the deadlines. At the foundation of scenario-based design lies the ability to identify scenarios, or recurring modes of operation with similar run time characteristics. There are two opposite ends to scenario identification. Some researchers have proposed techniques that, based on domain knowledge, identify hardware-independent scenarios in a media input stream. At the other end, other researchers have proposed techniques that identify hardware-dependent scenarios in a (semi-)automated way.

This paper proposes a scenario identification approach that bridges both opposite ends, and finds hardware-independent scenarios in an automated way. It does so by computing execution profiles on a per-frame basis that capture the application's code execution patterns. We find that Edge Vectors (EVs) are more accurate than Basic Block Vectors (BBVs) at capturing the variation in frame-level decode complexity. The complexity of the proposed automated scenario identification is comparable to existing hardware-dependent scenario identification approaches, yet the scenarios can be used across hardware implementations.

Categories and Subject Descriptors

C.3 [**Special-purpose and Application-based Systems**]: Real-time and embedded systems

General Terms

Design, Experimentation, Performance

Keywords

Video-decoding, Scenario-based design, DVFS

DAC 2008, June 8–13, 2008, Anaheim, California, USA.

1. INTRODUCTION

Energy consumption is a major design issue for most of today's systems. This is especially the case for battery-operated devices such as laptops, handheld computers, mobile phones, PDAs, etc. Media applications, such as video decoders, which are increasingly popular on most of these devices, typically have soft real-time requirements which can be exploited to save energy. Many researchers have proposed schemes that use dynamic voltage and frequency scaling (DVFS) and dynamic power management (DPM) for reducing energy consumption while still meeting the deadlines in media applications. DVFS-driven approaches for example exploit the time-varying decode complexity within a video stream by scaling down the voltage and frequency level when decoding a low-complexity frame; high-complexity frames that need more compute power to meet the deadline, are decoded at a higher voltage and frequency level.

At the foundation of these energy-efficient media decoding techniques lies the ability to identify so called *scenarios*, which group frames with similar decode complexity. Once the scenario is known that a frame belongs to, the appropriate voltage and frequency level can be installed for timely decoding the frame at reduced energy consumption.

There are basically two ways to scenario identification. At the one end of the spectrum, domain knowledge is used to identify hardware-independent scenarios [7, 8, 9, 16], i.e., an expert in the media application of interest identifies key media input stream characteristics that collectively give an accurate picture of a frame's decode complexity. Since the characteristics are based on the media stream only, they are independent of the underlying hardware on which the media stream is to be decoded. For example, counting the macroblocks and their types leads to an accurate characterization of the decode complexity of the H.264 AVC video decoder [7, 9, 16].

At the other end of the spectrum, hardware-dependent characterization determines a frame's decode complexity [4, 5, 6, 10, 11, 15]. This is typically done by comparing the decode time at a nominal clock frequency and voltage level, versus the per-frame deadline. A low-complexity frame results in a small decode time, whereas a high-complexity frame results in a large decode time. This hardware-dependent scenario identification approach is typically done in a (semi-)automated way, either online at run time or offline through profiling.

The goal of this paper is to explore and bridge the gap between hardware-independent scenario identification which requires expert intervention, versus (semi-)automated sce-

nario identification which yields hardware-dependent scenarios. In this paper, we combine the best of both worlds and propose an automated way of identifying hardware-independent scenarios. The novel method first characterizes all frames in a media stream database using Basic Block Vectors (BBVs) or Edge Vectors (EVs) which capture the decoder's executed basic blocks and edge counts, respectively, when decoding the given media stream. These BBVs or EVs then serve as input to cluster analysis which groups frames into scenarios. The media streams are annotated with scenario information which is then used at decode time to drive the DVFS-aware processor.

This paper makes two major contributions.

- We bridge the gap between expert-driven hardware-independent and automated hardware-dependent scenario identification, and show that we can identify hardware-independent scenarios in an automated way. These scenarios are nearly as effective as hardware-dependent scenarios and expert-driven hardware-independent scenarios.

- We show that characterizing frame-level behavior is more accurately done using edge vectors (EVs) than using basic block vectors (BBVs). This is an interesting result because prior work in program phase detection in general-purpose applications [19] found BBVs to be accurate for tracking the time-varying behavior in general-purpose applications, even at very large time granularities of hundreds of millions of instructions. This paper shows that BBVs do not accurately characterize the frame-level time-varying behavior in media applications — the reason is that different execution paths when decoding different macroblock types in the frame decode loop are lumped together in a BBV whereas an EV is able to capture these execution path differences.

2. SCENARIO-BASED DESIGN

There are two major flavors in the current literature of how to identify scenarios, which we discuss now in great detail.

2.1 Hardware-independent scenarios

In our prior work [7, 8], we present a scenario approach for media streams that uses domain knowledge to identify scenarios. The key idea of our proposal is to exploit the notion of frames with similar decode complexity both within and across media streams. An offline analysis determines frames across various media streams in a content provider's database that exhibit similar decode complexity, i.e., require similar compute power and energy consumption at decode time. This is done by computing a macroblock profile per frame for all media streams in the database. A *macroblock profile* counts the number of macroblocks of a given type in a frame [9, 16]. The purpose of a macroblock profile is to characterize the decode complexity in a hardware-independent way, i.e., a macroblock profile is independent of the decoder as well as the system on which the media stream is to be decoded, nevertheless, it provides a good picture of the frame decode complexity across hardware platforms.

Once macroblock profiles are collected for all media streams in the database, all frames can be represented as points in a multidimensional frame space, with one dimension per macroblock type. Cluster analysis [14] is then applied in the frame space to find groups of frames, called *scenarios*, based on their macroblock characteristics. The idea is that frames belonging to a given scenario show similar macroblock characteristics, and thus will likely represent similar decode complexity. Different scenarios show dissimilar macroblock characteristics, and will likely result in dissimilar decode complexity. Maintaining scenario identifiers can be done using a scenario database or using a separate scenario stream associated with each respective media stream [13].

In order to exploit the scenario information at decode time, the client needs to be profiled which is done by sending scenario representatives to the client. The client then decodes these scenario representatives and monitors (i) how long it takes to decode each scenario representative, and (ii) how much energy is consumed for decoding each scenario representative. Monitoring the decode time and consumed energy can be done using user accessible hardware performance counters that are available on modern microprocessors [12]. Upon decoding a media stream, the client reads the scenario information per frame, and then sets the appropriate voltage level and frequency through DVFS to reduce energy consumption while still meeting the deadlines [8].

2.2 Hardware-dependent scenarios

Most of the work on exploiting scenarios in media stream applications propose (semi-)automated scenario identification though, see for example [4, 5, 6, 10, 11, 15]. What all of these proposals have in common is that the scenarios are hardware-dependent, i.e., they are tied to one particular hardware platform and cannot be used across hardware platforms. The scenarios are typically identified by studying the number of cycles required to decode a frame, and frames with similar decode times are grouped to form a scenario. This profiling step can either be done at run time [4, 11, 15], or through an offline profiling step [5, 6, 10].

2.3 Discussion

An important advantage of hardware-dependent scenarios over hardware-independent scenarios is that the identification of the scenarios requires little or no domain knowledge. Hardware-dependent scenarios for a novel media decoding application can be identified automatically with little or no expert intervention. Approaches in hardware-independent scenarios on the other hand, require deep understanding of a novel media decoding application in order to identify the key frame features that will enable finding scenarios. For example, in [7, 8], knowledge is required about how a frame is built up from macroblocks. This is tedious and time-consuming to do for every media application of interest.

On the flip side, the disadvantage of hardware-dependent scenarios is that, by definition, scenarios need to be identified for each hardware platform of interest. For offline scenario identification, this may be very time-consuming in case multiple hardware platforms need to be supported. Online scenario identification does not have this disadvantage, however, identifying scenarios at run time may consume energy and/or may affect the perceived quality, i.e., it may result in deadline misses, as there typically is a learning phase during which scenarios are to be identified.

3. AUTOMATED HARDWARE-INDEPENDENT SCENARIO DETECTION

The goal of this work is to automate scenario identification while not giving up on the hardware-independence. By doing so, we aim at combining the best of both worlds which is to identify hardware-independent scenarios in a fully automated way without requiring domain expert interventions.

As a starting point, we consider a large set of media streams, and for each of these streams, we collect a per-frame profile. We consider two types of frame profiles. The first approach computes a **Basic Block Vector (BBV)** [18] per frame. A basic block is a linear sequence of instructions with one entry and one exit point. A Basic Block Vector (BBV) is a one-dimensional array with one element per static basic block in the program binary. Each BBV element captures how many times its corresponding basic block has been executed.

BBV example. Consider a program that executes the following dynamic basic block sequence 'ABABACABABACA', i.e., the program first executes the instructions in basic block A, followed by the instructions in basic block B, etc. The BBV looks as follows: $[A, B, C] = [7, 4, 2]$, i.e., basic block 'A' appears seven times during the dynamic execution, 'B' appears 4 times and 'C' appears 2 times.

The second approach computes an **Edge Vector (EV)** per frame. The edge vector (EV) is a one-dimensional array with one element per control flow edge in the program binary. A control flow edge is a transition (branch or jump) from one basic block to another basic block in the binary. Each EV element thus counts how many times its corresponding edge is taken. Edge counts contain more information than basic block counts, because the basic block counts can be derived from the edge counts, but not vice versa.

EV example. Consider the same example as above. The EV looks as follows: $[AB, BA, AC, CA] = [4, 4, 2, 2]$, i.e., the transitions between 'A' and 'B', and 'B' and 'A' appear 4 times in the dynamic instruction stream, and the transitions between 'A' and 'C', and 'C' and 'A' appear twice.

Scenario identification.

Once the BBVs or EVs are computed, cluster analysis is applied to group frames into scenarios. Cluster analysis [14] is a data analysis technique that groups n cases, in our case frames, based on the measurements of p variables, in our case the BBV or EV elements. There exist two commonly used types of clustering techniques, namely linkage clustering and K-means clustering. We advocate K-means clustering because it scales better with an increased number of cases. K-means clustering produces k clusters with the greatest possible distinction. The algorithm works as follows. In each iteration, the distance is calculated for each case to the center of each cluster. Each case is then assigned to its closest cluster and new cluster centers can be computed. This algorithm is iterated until no more changes are observed. Once the clustering is done, a representative can then be selected per scenario.

The end result of the proposed scenario identification methodology is k hardware-independent scenarios. These scenarios are identified in a fully automated way: computing BBVs and EVs, as well as the subsequent cluster analysis step, are fully automated. In addition, the whole process of identifying scenarios is not computationally expensive, and is comparable to the complexity of existing scenario identification approaches. Collecting the BBVs and EVs can be done through instrumentation or functional simulation; its complexity is proportional to the number of media streams in the content provider database, and their lengths. Cluster analysis using K-means clustering is efficient as well: it has a $O(kN)$ time complexity with N the number of frames to be clustered and k the number of clusters.

Window ROB/LSQ	32/16
Cache hierarchy	64KB L1 I/D-caches 1MB unified L2
Latencies (L1/L2/MEM)	L1: 2 cycles; L2: 20 cycles; MEM: 80ns
Branch predictor	hybrid 4K-entry tables, 10 cycle front-end pipeline
Processor width	4-wide
Functional units	4 integer ALUs, 2 memory ports

Table 1: Baseline processor model for this study.

Scenario exploitation.

Once the scenarios are identified, we need to annotate the media streams with scenario information, i.e., for each frame we need to determine the scenario it belongs to. This is done by collecting the BBV or EV for each frame, and locate the frame's BBV or EV in the scenario space. The scenario most similar to the BBV or EV — the scenario closest to the BBV or EV in scenario space — denotes the scenario the frame belongs to. The scenario annotation could be done in two ways. One possibility is that the scenario information is embedded in the video stream, i.e., the scenario identifier is added to the frame's header. This may be a viable solution if the scenario identifier can be encoded in the frame header in such a way that a non scenario-aware decoder can still decode scenario-aware media streams. A better and more practical solution is to maintain the scenario information in a separate scenario stream. The scenario stream can then be sent along with the video stream. The scenario stream solution is particularly interesting because it nicely fits within the container structure that is often used in multimedia streams, see for example the MP4 container format [13]. The amount of scenario information that needs to be sent from the content provider to the client is very limited, in theory no more than $\lceil log_2 N_s \rceil$ per interval with N_s the number of scenarios. In practice, this may require an additional byte, or in some implementations, for example in case the scenario information can be embedded in the already existing frame header format, communicating scenario information may not require sending any additional bytes at all.

At the decoder side, the scenario annotation can be used as for the hardware-independent scenarios described in section 2.1: the client is profiled and the client's optimal decode frequency and voltage level are determined per scenario; decoding a scenario-aware media stream then involves reading the frame's scenario and establishing the frame's optimal decode frequency and voltage level.

4. EXPERIMENTAL SETUP

Our experiments are done using the H.264 Advanced Video Coding (AVC) codec [17]. AVC is the new generation compression algorithm for consumer digital video. In our measurements we use version JM6.1 of the reference software of the JVT/AVC codec [20].

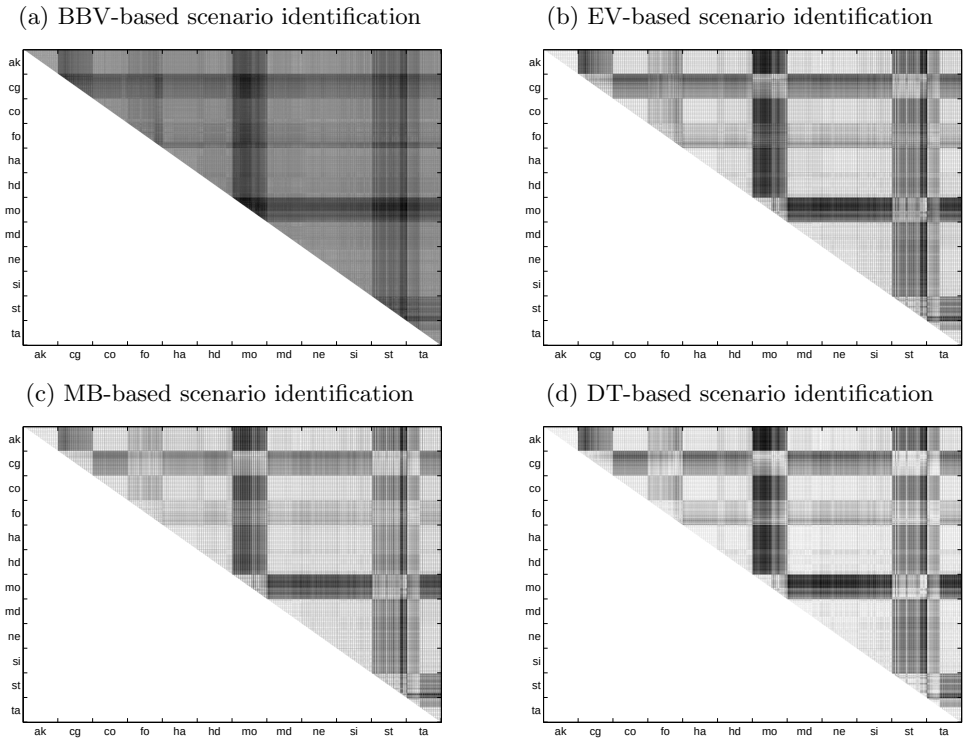

Figure 1: Similarity matrices for BBV, EV, MB and DT scenario identification.

In our evaluations we use twelve video sequences, each approximately 10 seconds at a decode rate of 30 fps. The results presented in this paper are obtained for these video streams in CIF resolution (352×288 pixels per frame). Further, we consider content-adaptive variable-length coding (CAVLC) and a GOP structure consisting of one I frame followed by 15 P frames.

The performance results presented in this paper as well as the BBVs and EVs were obtained using detailed cycle-level processor simulations using the SimpleScalar/Alpha v3.0 tool set [3]. Microprocessor energy consumption is estimated using Wattch v1.02 [1], assuming a 0.18um technology, 2V supply voltage, 0.5V threshold voltage and aggressive clock gating which shuts off unused parts of the microarchitecture while accounting for 10% leakage energy consumption. These simulation tools were extended to model frequency scaling as well as voltage scaling. When applying both frequency and voltage scaling we vary voltage with frequency based on $f \sim \frac{(V-V_{th})^{\alpha}}{V}$ [11] using 100MHz frequency steps over a range of 200MHz up to 2.7GHz. We also model the time cost for changing the processor operating frequency: 70 microseconds according to [2]; this inter-scenario switching cost is much shorter than the inter-frame deadline (30 milliseconds). The baseline processor model used in this paper is a contemporary 4-wide superscalar microarchitecture, see Table 1 — we target high-performance embedded systems.

Note that in all of our experiments, we assume a leave-one-out methodology. This means that when evaluating the efficacy of a given scenario identification approach for a given video stream we leave that video stream out of the content provider's database for building the scenarios. This reflects what is to be expected in practice whenever a new video stream is added to the content provider's database.

5. EVALUATION

In the evaluation, we consider four ways of identifying frame-level scenarios:

- BBV-based: A frame is characterized through a Basic Block Vector (BBV).
- EV-based: An Edge Vector (EV) characterizes a frame.
- MB-based: A Macroblock Profile (MB) characterizes a frame — this corresponds to domain knowledge based hardware-independent scenario identification, see Section 2.1.
- DT-based: A frame is characterized by computing the Decode Time on a particular hardware platform — this corresponds to the (semi-)automated hardware-dependent scenario identification approach, see Section 2.2.

5.1 Similarity

Before evaluating these four scenario identification approaches in terms of energy reduction and missed deadlines, we first evaluate the efficacy of these approaches in terms of their ability to discern decode complexity differences. For doing so, we compute a similarity matrix. A similarity matrix is an upper triangular $N \times N$ matrix with N the number of frames in the media stream database. An entry at position (x, y) represents the (normalized) Manhattan distance between frame x and frame y. The distance between two frames is computed using their respective BBVs, EVs, MBs or DTs. The gray scale represents the distance between two frames: black means completely different and white means no difference.

Figure 1 shows similarity matrices for the four approaches. These similarity matrices provide insight into how (dis)similar

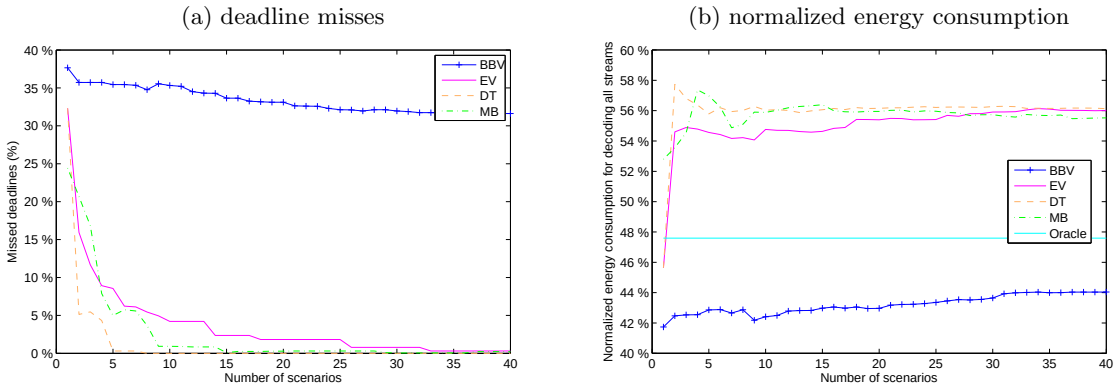

Figure 2: Average fraction deadline misses (left) and average normalized energy consumption (right) as a function of the number of scenarios.

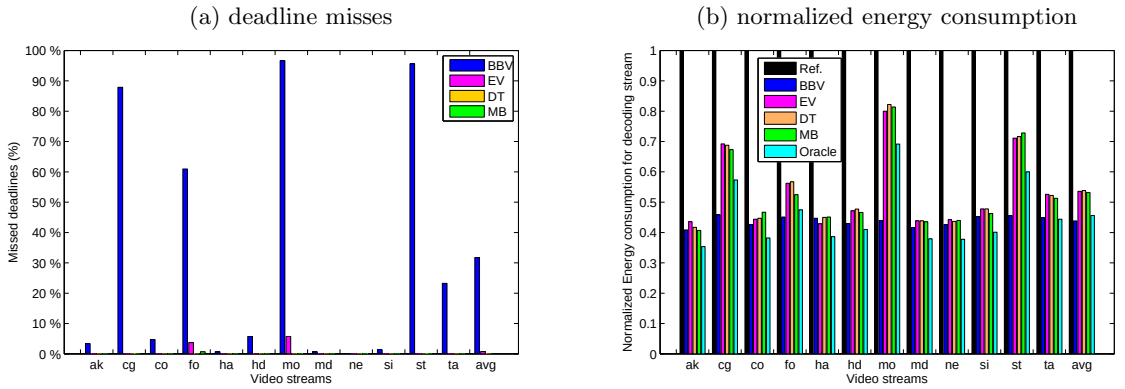

Figure 3: Fraction missed deadlines per benchmark (left) and normalized energy consumption (right) assuming 32 scenarios.

video streams are in terms of their decode complexity. For example, the mobile video stream is dissimilar from all other video streams except for coast guard, foreman and stefan — see Figure 1(d) which shows black bars (dissimilar decode complexity) for mobile with all other video streams except for the white/gray squares with coast guard, foreman and stefan. From Figure 1, we observe that the EV and MB approaches are much better at discriminating frame decode complexity differences than the BBV approach: there is more contrast in the similarity matrices for the EV and MB approaches than for the BBV approach, and the EV and MB similarity matrices resemble the similarity matrix of the DT approach much better.

The reason why EVs outperform BBVs is that EVs capture execution path differences resulting from macroblock differences in the frame decode loop, whereas BBVs average the various execution path differences. This is an interesting result because BBVs were shown to be accurate at tracking the time-varying program execution behavior of general-purpose applications [19], even at the granularity of hundreds of millions of instructions. For media applications, with a frame decode loop of on the order of a hundred million of instructions, however, BBVs seem to be inaccurate.

5.2 Missed deadlines versus energy reduction

We now evaluate the proposed BBV and EV-based scenario identification approaches in terms of missed deadlines and energy reduction when deployed to a DVFS-aware processor system. Figure 2 shows the fraction of missed deadlines and the normalized energy consumption as a function of the number of scenarios, averaged across all media streams — these graphs show the trade-off between energy consumption reduction and missed deadlines as a function of the number of scenarios. Figure 3 shows per-benchmark results assuming 32 scenarios. We observe that EV and MB are nearly as effective as DT, whereas BBV is significantly worse. The BBV approach results in a substantial amount of deadline misses, and for some media streams, nearly all frame deadlines are missed. This confirms our earlier finding that BBVs are unable to capture a frame's decode complexity. EVs on the other hand, are as effective as MB and DT when considering a large enough number of scenarios. For a small number of scenarios, DT performs better, especially with respect to the number of deadline misses; this advantage reduces with a increasing number of scenarios. EV-based scenario identification results in scenarios that are close to what can be obtained using oracle scenario information in which each frame gets decoded at its optimal clock frequency that minimizes energy consumption while meeting the deadlines. This results in an average 44% reduction in energy consumption compared to our point of reference which decodes all media streams at a fixed clock frequency so that all frames get decoded within their deadlines.

5.3 Resource prediction

In our prior work [7, 8], we also used hardware-independent MB-based scenarios to predict the required resources, such as energy consumption, for decoding a media stream at a given QoS level. Figure 4 shows the prediction error in predicting the energy consumption using the EV, BBV and

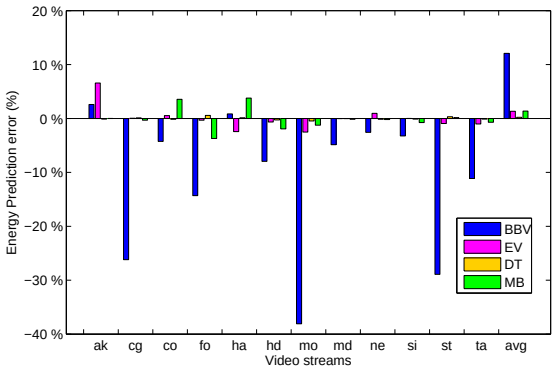

Figure 4: Energy consumption prediction error per benchmark assuming 32 scenarios.

DT-based approaches and compares their accuracy against the MB-based approach. The prediction error using EVs is nearly as accurate as using MBs with an average error of only 2%. BBVs on the other hand, result in prediction errors up to 38%.

6. CONCLUSION

A scenario-based design methodology exploits the time-varying behavior observed in media applications by grouping input stream frames into scenarios, and by adapting the system on a per-scenario basis. By doing so, energy consumption can be reduced while meeting soft real-time constraints. The current literature describes two basic approaches to scenario identification. At the one end, hardware-independent scenarios are identified using domain knowledge. At the other end, hardware-dependent scenarios are identified (semi-)automatically.

This paper bridged the gap between both ends and proposed Edge Vectors as an accurate way of characterizing the frame-level decode complexity in a hardware-independent manner. Basic Block Vectors on the other hand, are unable to identify execution path differences in the main frame decode loop resulting from the variety of macroblock types in a frame. Scenario identification using Edge Vectors can be automated so that the scenario-based design methodology can be easily applied to other media applications of interest.

Acknowledgements

Juan Hamers is supported by a BOF grant from Ghent University. Lieven Eeckhout is a Postdoctoral Fellow with the Fund for Scientific Research–Flanders (Belgium) (FWO–Vlaanderen). This research is also partly supported by the FWO projects G.0160.02 and G.0255.08, and HiPEAC.

7. REFERENCES

[1] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *ISCA*, pages 83–94, June 2000.

[2] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen. A dynamic voltage scaled microprocessor system. *IEEE Journal of Solid-State Circuits*, 35(11):1571–1580, Nov. 2000.

[3] D. C. Burger and T. M. Austin. The SimpleScalar Tool Set. Computer Architecture News, 1997. See also `http://www.simplescalar.com` for more information.

[4] K. Choi, K. Dantu, W.-C. Cheng, and M. Pedram. Frame-based dynamic voltage and frequency scaling for a MPEG decoder. In *ICCAD*, pages 732–737, Nov. 2002.

[5] E.-Y. Chung, L. Benini, and G. De Micheli. Contents provider-assisted dynamic voltage scaling for low energy multimedia applications. In *ISLPED*, pages 42–47, Aug. 2002.

[6] S. V. Gheorghita, T. Basten, and H. Corporaal. Profiling driven scenario detection and prediction for multimedia applications. In *IC-SAMOS*, pages 63–70, July 2006.

[7] J. Hamers and L. Eeckhout. Resource prediction for media stream decoding. In *DATE*, pages 594–599, Apr. 2007.

[8] J. Hamers and L. Eeckhout. Exploiting media stream similarity for energy-efficient decoding and resource prediction. *ACM Transaction on Embedded Computing Systems (TECS)*, 2008. To appear.

[9] M. Horowitz, A. Joch, F. Kossentini, and A. Hallapuro. H.264/AVC baseline profile decoder complexity analysis. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):704–716, July 2003.

[10] Y. Huang, S. Chakraborty, and Y. Wang. Using offline bitstream analysis for power-aware video decoding in portable devices. In *MM*, pages 299–302, Nov. 2005.

[11] C. J. Hughes, J. Srinivasan, and S. V. Adve. Saving energy with architectural and frequency adaptations for multimedia applications. In *MICRO*, pages 250–261, Dec. 2001.

[12] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *MICRO*, pages 93–104, Dec. 2003.

[13] ISO/IEC. Information technology – coding of audio-visual objects – part 14: Mp4 file format. ISO/IEC 14496-14:2003, Apr. 2004.

[14] R. A. Johnson and D. W. Wichern. *Applied Multivariate Statistical Analysis*. Prentice Hall, fifth edition, 2002.

[15] Z. Lu, J. Hein, M. Humphrey, M. Stan, J. Lach, and K. Skadron. Control-theoretic dynamic frequency and voltage scaling for multimedia workloads. In *CASES*, pages 156–163, Oct. 2002.

[16] M. Mattavelli and S. Brunetton. Implementing real-time video decoding on multimedia processors by complexity prediction techniques. *IEEE Transactions on Consumer Electronics*, 44(3):760–767, Aug. 1998.

[17] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi. Video coding with H.264/AVC: Tools, performance and complexity. *IEEE Circuits and Systems Magazine*, 4(1):7–28, Jan. 2004.

[18] T. Sherwood, E. Perelman, and B. Calder. Basic block distribution analysis to find periodic behavior and simulation points in applications. In *PACT*, 3–14, Sept. 2001.

[19] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *ASPLOS*, pages 45–57, Oct. 2002.

[20] K. Sühring. H.264/AVC reference software. `http://iphome.hhi.de/suehring/tml/download/`.