Control strategies for H.264 video decoding under resources constraints*

Anne-Marie Alt
Inria Grenoble Rhône-Alpes, NeCS project-team
Montbonnot, Inovallée
38334 St Ismier Cedex, France
Anne-Marie.Alt@inrialpes.fr

Daniel Simon
Inria Grenoble Rhône-Alpes, NeCS project-team
Montbonnot, Inovallée
38334 St Ismier Cedex, France
Daniel.Simon@inrialpes.fr

ABSTRACT
Automatic control appears to be an enabling technology to handle both the performance dispersion in highly integrated chips and computing power adaptability under varying loads and energy storage constraints. This work in progress paper presents a case study, where a video decoder is controlled via quality loops and frequency scaling, to meet end-users requirements mixing quality and energy consumption related constraints.

Categories and Subject Descriptors
H.4 [**Information Systems Applications**]: Miscellaneous—*Adaptive scheduling*

General Terms
Design,Experimentation

Keywords
Feedback scheduling, QoS, H.264 video decoder

1. CASE STUDY OVERVIEW
Mobile devices such as PDAs and mobile phone are increasingly integrating multimedia and telecommunication embedded applications, thus requiring increasing on-board computing power. The required high computing capacities can be provided by highly integrated multi-core chips using several tenths of micro-controllers, as forecast by the Aravis (Advanced Reconfigurable and Asynchronous Architecture for Video and Software radio Integrated on chip) project.

Such chips of the future will integrate extremely small scale CMOS manufacturing, e.g. silicon foundries currently target 32 nm or even smaller gates. A nasty consequence of very high integration is the silicon process dispersion, i.e. cores factored on the same chip will behave differently. In particular different cores will be capable of different maximum clock frequencies even if coming from the same design [5]. To take full benefit from the potentially available computing power, it is needed that each computing node (or group of nodes) can be driven up-to its maximum clock frequency, thus making the overall computing architecture heterogeneous and globally asynchronous. In practice several nodes sharing a common frequency domain are gathered in a *cluster*, and clusters working at different frequencies are linked via an Asynchronous Network on Chip.

*This work is funded by Minalogic within the Aravis project http://www.minalogic.com/projets.htm

1.1 QoS requirements
These highly integrated chips are expected to be used in many computing intensive fields. Typical ones are multimedia applications, such as receiving, decoding and displaying high definition television streams on mobile devices : this particular application provides the case study described in the paper. Thanks to the important embedded computational power many functions that were traditionally hardwired can be now implemented in software. This is for example the case for the radio communication receiving system, where components like filters, mixers and codecs can be now implemented with increased flexibility inside the programmable components of a software defined radio system (SDR). The extra computing power can also be used to decode video streams with high definition quality.

However high computing power has drawbacks in term of energy consumption, especially for the case of mobile devices with limited embedded energy stored in a battery. Hence a trade-off between a measure of the multimedia application quality, the available on-board energy and the desired time to battery exhaustion must be managed, which may be translated into a control problem. Although not formally defined up-to-now this problem might be stated, among other formulations, as “optimization of the decoding quality under energy consumption constraint”.

Trading performances and resources is relevant of Quality of Services (QoS) problems, which were primarily stated and studied in the framework of networking. More generally, and beyond the initial (network related) meaning, QoS may be viewed as the management of some complex quality measures, assumed to provide an image of the application requirements satisfaction. Indeed QoS problem statements have been already used for energy aware multimedia applications, e.g. [1], most often focusing on networking load and communication management rather than on computing itself.

However an approach stating video processing as a QoS problem is reported in [9]. The QoS is evaluated via end-users perception criteria and enables tuning decoding parameters such as picture quality, deadline misses and quality level switches. Scalable processing is provided using several quality levels for frames decoding, each one associated with a corresponding computing cost. Video processing is known to be subject to fluctuations and content-dependent processing times : at run-time QoS management is made adaptive and robust against the incoming stream uncertainties by an active closed-loop control, where measures of the system’s outputs (e.g. actual CPU load and deadlines miss) are fed back to the decoder’s input to chose the next frame decoding quality level. Even

if considering a single CPU with constant processing power the approach shows a very effective and flexible decoding adaptation capability.

Such closed-loop control scheme is currently considered in the framework of the Aravis project, and follows several steps. Control design first needs a definition of the control objectives and an analysis and modeling of the process to be controlled. Basically, control uses an error signal between the desired and the measured (or estimated) state or output of the system. The output signals which are significant for control purpose must be identified and the corresponding sensors must be implemented. Then various control algorithms can be used to cyclically compute commands to be applied to the process via actuators, which must be also identified and implemented.

2. H.264/SVC CODEC OVERVIEW

H.264 (also known as MPEG-4/AVC) is an international video coding standard for Advanced Video Coding proposed by the Joint Video Team : it was first approved in 2003 [8]. It is intended to be used in many multimedia applications such as downloading and streaming via Internet, software defined radio, multimedia for mobile devices and high definition television. More recently the Scalable Video Coding extension (SVC) has been defined to provide scalability capabilities, e.g. enabling multiple resolutions and various quality levels in compressed bitstreams. Therefore this extension is expected to provide the actuators needed by a QoS control loop working on top of the decoder.

2.1 Features of H.264/SVC

The H.264/SVC reference software, which implement the features of the Joint Scalable Video Model (JSVM) [1] algorithm, has been elected for preliminary experiments. Although it is not optimized, this software implements all the features defined for the H.264 standard and for the associated SVC extension which are detailed in [7].

Basically SVC enables to decode only some selected parts of the incoming compressed bitstream. Figure 1 below sketches the coding/decoding process. First, the raw input video flow is encoded to obtain a compressed bitstream. At coding time SVC allows for encoding the input video with combinations of different temporal rates, spatial resolutions and quantization steps. At the output of the encoder, the bitstream which contains several quality layers is sent to the decoder via a communication medium. Before decoding, selected partial bitstreams are extracted from the initial bitstream. Finally, only selected partial bitstreams are decoded, while switching between layers is possible in some cases.

Figure 1: Video Transmission

Three types of scalability are allowed by SVC:

- Spatial scalability enables to encode a video with several resolutions (i.e. number of pixels in a picture). The original high resolution video is down-converted to new video streams with lower resolutions. The final bitstream contains the video with all the encoded resolutions. The decoder decodes first the picture with the lowest resolution, then pictures with higher resolutions if needed.
- Temporal scalability enables to encode all or a part of the frames of the original video with different rates. For example it can be chosen to encode only half of frames to save computing power or networking bandwidth: in that case the displayed video only contains 15 frames per second rather 30 frames per second in the original video stream.
- Quality scalability allows to encode the frames with several quantization steps. The quantization step selectively cancels some information from the original video: its effect can be compared to a low-pass filter. Indeed, the human eye is more sensible to the low frequencies than high frequencies, so that canceling high contrasts can be done progressively with a moderate impact on the visual perception. High quantization steps lead to lower quality pictures but also to lower computing costs. For a frame containing several quantization based quality layers, the decoder must process first the lowest quality layer (with the highest quantization step), then layers of increasing quality.

These different scalability properties can be combined to encode/decode a video stream as depicted in Figure 2. It is stated that the decoding process necessarily flows from lower to higher quality layers. Obviously all the quality layers needed by the decoder must have been previously encoded and transmitted over a communication channel to the decoder.

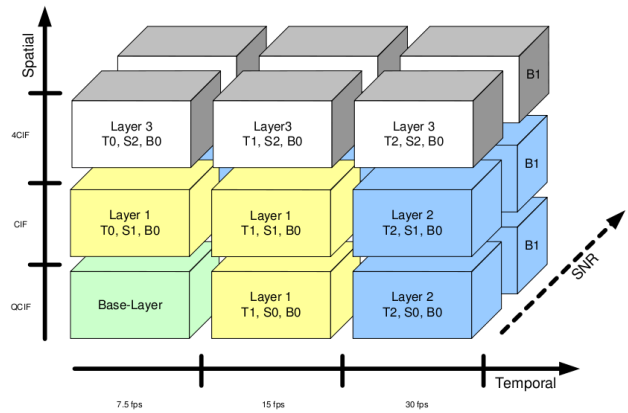

Figure 2: Bitstream with spatial, temporal and quality layers

2.2 Video bitstream structure:

A video sequence is made of 3 types of pictures: I pictures are references pictures which are encoded independently of any other, P pictures are encoded using the previously encoded I picture, and B pictures are encoded using both the I and P pictures of there patterns. Hence the order in which pictures are displayed is different from the order of the pictures encoding/decoding. For instance, if the display order is IBBBPBBBI then the encoding/decoding order will be IPBBBIBBB. The IPB pattern is defined at coding time and is invariant along all the video stream. The constant interval between successive I pictures is the *intra period* of the pattern and the number of P pictures between two I is also constant and known as the *group of pictures*.

[1] http://ip.hhi.de/imagecom_G1/savce/downloads/SVC-Reference-Software.htm

The bitstream is made of so-called access units (which contain each one picture). Each access unit is divided in NAL (Network Abstraction Layer) Units of two types: *slices* (or Video Coding Layer Units) which contain pixels related information, and non-VCL Units which contain access units structure related information such as the slices parameters. The number, size and shape of the slices inside access units are defined at coding time and constant over all the bitstream. Slices are themselves divided in macro-blocks.

2.3 Decoding process

The present work only focuses on the decoder which is now detailed through an example. Let us decode a bitstream which contains a video with two resolution levels: 352x288 pixels and 704x576 pixels. Each resolution layer of the video contains two quantization layers (Qp) 30 and 20 (recall that higher is the quantization step, lower is the quality). Therefore the bitstream contains 4 layers, sorted from the worst to the best quality:

- layer 0: resolution 352x288 pixels and Qp =30
- layer 1: resolution 352x288 pixels and Qp =20
- layer 2: resolution 704x576 pixels and Qp =30
- layer 3: resolution 704x576 pixels and Qp =20

Note that this ordering fits with the ordering given sorting the Peak Signal to Noise Ratio of the compressed frames, referenced to the original raw frame.

The bitstream is necessarily decoded in this order, from the worst to the best quality layers. Switching between resolution layers is only possible when decoding I pictures, and the P and B pictures can be decoded only with the decoded resolution and quantization layers decoded for their I reference picture (but they can be decoded with a quantization lower than their I reference) [7].

The decoding process for each layer and for each slice follows the following steps, which are identical for all slices:

- initialization of the slices and decoding parameters;
- slice parsing, analysis of the bitstream and entropy decoding. Entropy decoding consists in convert the binary information in decimal coefficients corresponding to each macro-block.
- slice decoding: this step reconstructs the picture thanks to the decimal coefficients computed in the entropy decoding.
- optional final processing using a loop filter. This filter is applied to improve the final quality of the picture and is executed at the end of the total frame decoding process.

3. PARALLELIZATION STRATEGIES

Slices are processed separately and identically. As there is no interaction between individual slice decoding processes, slices decoding can be performed sequentially or in parallel according to the system at hand. In the sequel the decoder is assumed to be executed on cluster made of P identical processors, all working at the same (possibly variable) speed. Three possible strategies for the decoding parallelization have been considered (assuming that frames are divided in S slices):

Picture-level parallelism: due to the dependency between some pictures the order in which they are decoded is not free. Inside a pattern the I picture must be processed first, then the P ones, and finally the B frames, which can be processed in parallel on some processors while the I picture of the next pattern begins its processing. The effectiveness of this method highly depends on a good matching between the number of available processors and the bitstream pattern structure, and in a general case the amount of time savings due to parallel processing is limited by the dependencies relations between pictures.

Slice-level parallelism: each picture is divided into slices, which are separately decoded each on one processor of the cluster. Recall that the slices map is defined at coding time and is constant for all the bitstream. Ideally the number of slices matches the number of processors. After parsing the picture each slice undergoes all decoding steps, finally the decoded parts are sewed up again and the picture is optionally filtered and post processed. When the number of slices does not match the number of processing nodes variants of this strategy can be implemented: in particular if P < S, groups of P slices can be decoded sequentially, and if P > S, pictures can be mapped on S nodes and several frames can be decoded in parallel (taking into account dependencies). As the slice map is known when decoding the first frames the mapping between slices and nodes can be chosen in pre-defined configurations when decoding begins.

Macro-block-level parallelism: each slice contains many macro-blocks which decoding order must be respected due to dependency constraints. The first macro-blocks must be decoded sequentially, then the following independents macro-blocks can be decoded in parallel until joining a synchronization between blocks, and the process is repeated until the end of the slice decoding. This strategy is independent from the bitstream configuration but the number of steps to be sequentially executed (entropy decoding) and of midway synchronizations severely limit the potential parallelism and associated time savings. [2].

The second strategy has been chosen because it provides the best parallelism level and potential speed-up. The *libasync* parallelizing tool [10] associated to the event-based programming model [6] is currently used to develop a parallel version of the H.264/SVC decoder in which the actual number of cores is a tunable parameter $\in \{1, 2, 4, .., 2^n, ..\}$. Reading of the first access unit gives the structure of the slices map in frames and allows for actually mapping the incoming bitstream onto the hardware architecture. Note that in this case study parallelization is only a way to speed-up the decoding process and is not considered for on-line re-mapping under feedback decisions.

4. FEEDBACK SCHEDULING SETUP

Preliminary control design is sketched in this section. Control design needs sensors to observe the controlled process and actuators to modify its state and output. In the particular case of feedback control of computing systems, sensors are provided by software probes used to build on-line indicators carried out the processing activity. Actuators are provided by function activation, parameters tuning, or processing suspend and resume under control of an operating system. As the bitstream decoding quality is the object of control, models of the decoder quality (i.e. the controller performance) as a function of various execution parameters (desired quality levels) are estimated from experiments. These models will be further used to formalize the control objectives, e.g. using a QoS formulation. Besides data coming from the reported experiments many ideas and assumptions are inspired by the work described in [9].

4.1 Control Strategies

Ideally the goal of the controller is to maximize the quality of the displayed video stream under constraint of energy consumption. Hence, according to an available computing budget, the video must be decoded with the lower possible quantization step, higher resolution and maximum rate. The allowed computing budget itself depends on the available on-board energy storage and desired operating life. This high level controller works with long term objectives with a time scale slow compared with the time scale of frames processing. At the lower level, decoding frames has basic deadlines related to video display, typically 40 msecs for frames displayed at standard television rate. However the decoding computing load is subject to fluctuations due to the varying content-dependent computation duty of the successive frames. Therefore an on-line adaptation of the decoding parameters (quality layers) can be associated with the frequency scaling capabilities of the cluster to meet the requested video rate.

These various control objectives and timing scales lead to define a hierarchy of two control loops to manage the decoding quality (Figure 3). At high level a QoS controller manages the application performance according to the available resources and end-user's requirements. At a lower level the frame controller works at the pictures stream time scale and tightly cooperates with the processing speed controller integrated in the cluster.

4.2 Control hierarchy

As usual the design of control loops needs to define a control architecture together with the selection of the set of sensors and actuators to be used. From top (application software and long terms objectives) to bottom (silicon level and high control rate), the control hierarchy is (as depicted by Figure 3):

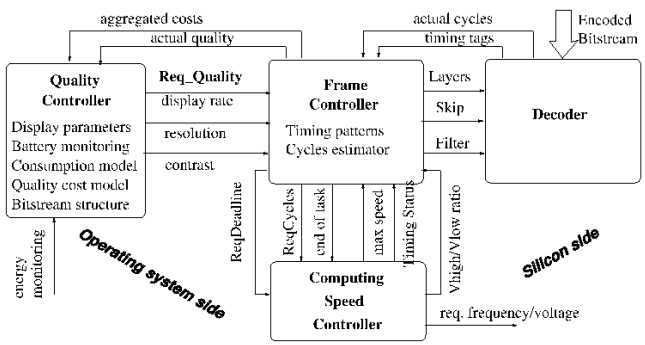

Figure 3: Control architecture

- The Quality Controller (Q-C) software runs in a master processor on top of the Operating System. It communicates with the clusters via an Asynchronous Network on Chip (ANOC).
- The Frame Controller (F-C) software runs in one of the nodes of the considered cluster as a high priority task.
- The Computing Speed Controller (S-C) is integrated in the cluster's silicon, together with the voltage/frequency controllers.

4.3 Quality controller

This controller manages long terms and user's defined goals. Informally the control objective consists in trading-off the decoded bitstream quality and the energy storage lifetime, with an average energy consumption level in mind. Quality parameters are expressed in term of display requirements, e.g. HD or standard mode, display rate and screen resolution. The end-user's may select different weights between these parameters, e.g. by imposing a high definition display whatever the cost, or asking for a mandatory lifetime before refill.

Using a rough model between the quality layer and energy costs, this first loop aims at setting the current requested quality level to be decoded. The on-line monitoring of the battery level and voltage decaying rate are fed back for on-line estimation and correction of this quality set point. Thanks to the cluster speed controller in [4], the relations between the cluster's computing speed and the electrical power needed to feed the cluster can be approximated by monotonic functions. In other words, higher is the demanded computation burden, higher is the energy consumption: it is expected that such monotonic cost functions lead to a simple control design, even if a formal statement for this control problem remains to be done.

Quality layers and computing costs. After decoding the first frames, the structures of the bitstream (number of quantization/resolution layers, IPB structure, pictures rate and slices map) are known and can be used to actually set the decoding parameters. Some of the video parameters are constrained by the incoming bitstream and by the display mode:

- the display rate constraints the average deadline for each picture (e.g. 40 ms for standard TV rate);
- the quantization and resolution layers in the decoded bitstream must be encoded in the source stream;
- B and P frames cannot be decoded at a quality higher than the one of their reference I frames.

Switching on-line the display rate should be avoided as far as possible due to the visible effect on the display. Hence the usual choice for the variable decoding parameters, able to handle varying computing capabilities, is the requested quality layer to be decoded. A correct estimation of the quality set point needs the knowledge of a model (cost function) to link up the quality layers and computing loads (and at least to understand their respective variations).

Figure 4 plots average cycles number to decode the five quantization layers of a particular bitstream. The quantization steps are here equidistant and set to QP = 40, 34, 28, 22, and 16. From the left to the right the plot shows the average cycles number for I, P and B frames. (It is assumed that measures taken from a fixed frequency CPU provide a good image of the computing load in term of statements to be executed).

Figure 5 shows cycles number for a bitstream made of a mix of resolution and quantization layers with the parameters already given in section 2.3.

These experiments show that the choice of the quantization or resolution layers has a significant impact on the computational load, and that switching resolution has a rough influence while quantization may provide fine control.

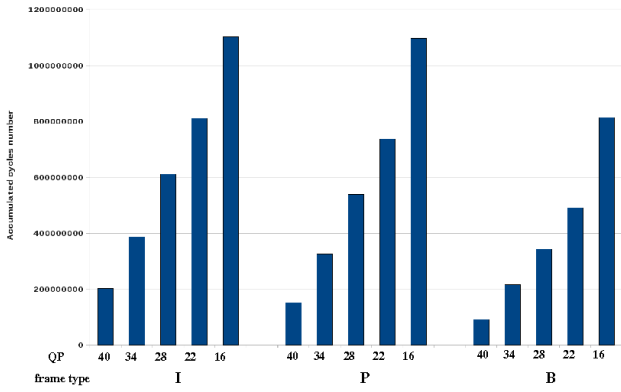

Figure 4: Computation load for Quantization layers

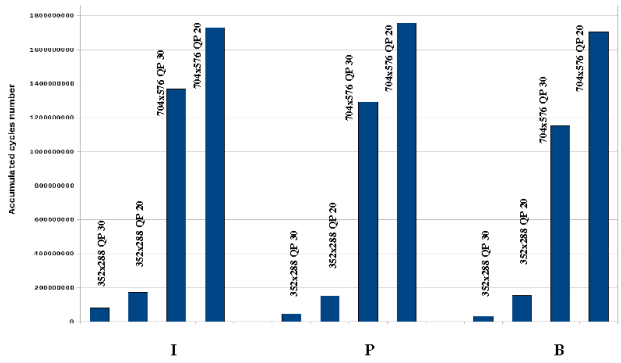

Figure 5: Computation load for combined Resolution-Quantization layers

Therefore decoding only the lower quality layers appears to be an effective actuator to reduce the decoding cost and the related energy consumption, and to manage the trade off between the displayed quality and the energy consumption constraints. Other preliminary experiments, using an elementary controller to skip high quality quantization steps in case of overload, showed that switching between quantization layers has a very moderate impact on the viewer's perception, while efficiently saving execution cycles and avoiding deadlines overshoots.

Recall that the quality layers contained in the incoming bitstream must be decoded in sequence, with the low quality layers first: observing the latter figures it appears that increasing the allocated computation load monotonically increases the decoding quality: once again this nice property is expected to help the design of the quality controller.

4.4 Frame controller

This controller feeds the computing speed controller with estimations of the amount of computations to be performed within an associated deadline.

Typically deadlines are associated with the video rate, e.g. 40 msecs are allocated to fully decode and display one image. However, even if the display video rate must be respected as far as possible, there are no strong synchronous constraints between the video source capture, encoding, decoding and display: latencies equivalent to several frames can be allowed, hence there is room for scheduling flexibility at decoding time.

Recall that, due to dependencies between images of different types I, P and B, the displayed order is different than the decoding order, so that the displayed flow is inevitably delayed w.r.t. the incoming bitstream. Following the ideas in [9], an additional buffer is added to the frames decoding and display queue, so that decoding is performed several frames ahead of display. This added buffer is used to give space and accommodate for the varying computing loads between frames. Measurements of decoding execution times were made to evaluate the profile and amplitude of computing load variations along a movie. Execution times measured from a 1000 frames long movie have been sorted according to the frame type (I,P or B) on Figure 6 where the bitstream has a unique layer with 624x352 pixels resolution and quantization step 28. This video sequence contains a mix of quiet and action plans.

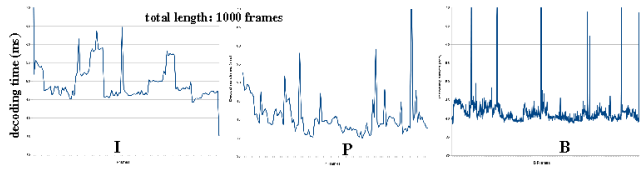

Figure 6: Decoding times for I, P and B frames

It can be observed, especially for the reference I frames, that the decoding times are almost constants for quite long intervals, with abrupt changes between flat areas, and isolated high values. The observation of constant intervals enables to estimate the cycles number with sufficient accuracy. The maximum value for theses isolated peaks suggests that a 3 frames deep control buffer would be able to damp most of the computing load variations.

The main goal of the F-C is to provide the underlying S-C with estimates of the number of cycles $\hat{q}_{k+1}$ to be processed for the next frame f_{k+1} within a $d_{r_{k+1}}$ requested deadline. According to [9] and to the measures plotted in Figure 6, an estimate of $\hat{q}_{k+1}$ can be often taken as the last q_k reported by the S-C, or by filtered values of the last executions, e.g. $\hat{q}_{k+1} = (1-\alpha)q_k + \alpha q_{k-1}$.

Considering a fixed ideal schedule $\{..., t_{k-1}, t_k, t_{k+1}, ...\}$, e.g. with equidistant interval of 40 ms, a first basic feedback loop is aimed to regulate the requested deadlines d_{r_k} to their ideal value t_k. Due to the rough prediction of the computing load $\hat{q}_k$ the actual (measured) deadline is $d_k = d_{r_k} + \delta_k$. Assuming that the computing load is almost constant, the computing overshoot can be driven to 0 according to $\delta_{k+1} = (1-\beta)\delta_k$ with $0 < \beta < 1$, leading to the elementary deadline controller:

$$d_{k+1} = t_{k+1} + (1-\beta)\delta_k$$

Indeed this control loop may accommodate for short term variations of the computing load for each frame. It capabilities can be exhausted in case of several successive peak loads able to overflow the 3 frames ahead buffer. In that case only I frames will be fully decoded up-to the requested quality level, and the depending P and B frames decoding can be truncated up-to recovering enough buffer space. Thanks to the signals that are fed back by the decoder and by the computing speed controller several control decisions can be considered, sorted by their expected increasing impact on the dis-

played quality:

- truncation of the decoding process at a quality layer lower than the set point can be done at any point for B and P frames;
- a comparison between a reference decoding timing pattern and actual tags inserted in the decoder may help to anticipate overloads and abort useless steps rather than awaiting a deadline miss: in particular the final filtering action can be skipped;
- in case of accumulated overload peaks running beyond nominal control actions, skipping or aborting a frame decoding may be taken as an emergency action, allowing to reset the decoding stack. This action must be as far as possible avoided especially for I frames.

From a control point of view, the decoding process has a simple dynamics (mainly due to measuring and averaging filters). Therefore it is expected that simple control design holds (as in similar referenced works) and that stability will not be difficult to assess. However the adequate tuning of these control strategies, e.g. filters damping and threshold values, needs further experiments and a more formal characterization of signals patterns and control objectives.

4.5 Computing speed controller

A computing speed controller is implemented in each cluster. Its inputs are an image of the computing power needed by the application software: this set point is cyclically given by the frame controller as pairs $< nb\_cycles, deadline >$ which correspond to computation units. It outputs voltage and frequency set points forwarded to the lowest levels V_{DD} hopping voltage converter and programmable ring oscillator. This controller is designed to minimize the energy needed to execute the given amount of computations: therefore it minimizes the time spent when the cluster works at high voltage. It is fed back by the actual number of executed statements so that it can compensate for process variability to finish the requested computations just on time, and may use clock gating to stop the activity of idle nodes.

During a computing activity it also records information about the cluster's state and computation progress. In particular two signals, which appear to be relevant for frame control anticipation, are fed back to the frame controller: the High/Low voltage ratio observed during the last frame processing gives an image of the safety margin before missing a deadline, and the deadlines and sub-deadlines missing values allows for anticipating processing overloads.

This controller is generic and implemented in silicon. Details about its design can be found in [4] for design basics and single core control, and in [3] for the cluster version.

5. SUMMARY AND FURTHER WORK

In this work in progress paper, it is conjectured that a hierarchy of control loops would efficiently manage the trade-off between a multimedia application quality index and computing resources usage. Indeed control design usually starts with process modeling, based (when possible) on the process internals analysis, or on input/output relations analysis. Control loops basically use sensors and actuators, which must be carefully selected, implemented and calibrated to allow for effective feedback and control actions. The current study focused on the identification and implementation of the sensing and actuating devices, while control design is currently only sketched. More detailed control design and experimentation are expected to be available by the venue of the workshop.

6. ACKNOWLEDGMENTS

The authors gratefully acknowledge Fabien Mottet (INRIA Sardes team) for his assistance during the decoder parallelizing design and coding.

7. REFERENCES

[1] J.-C. Chiang, H.-F. Lo, and W.-T. Lee. Scalable video coding of H.264/AVC video streaming with QoS-based active dropping in 802.16e networks. In *22nd Int. Conf. on Advanced Information Networking and Applications*, Okinawa, Japan, 2008.

[2] J. Chong, N. Satish, B. Catanzaro, K. Ravindran, and K. Keutzer. Efficient parallelization of H.264 decoding with macro block level scheduling. In *IEEE International Conference on Multimedia and Expo, ICME'07*, Beijing, China, July 2007.

[3] S. Durand and N. Marchand. Energy consumption reduction with low computational needs in multicore systems with energy-performance tradeoff. In *48th IEEE Conference on decision and control CDC'09*, Shanghai, China, Dec. 2009.

[4] S. Durand and N. Marchand. Fast predictive control of micro controller´s energy-performance trade-off. In *3rd IEEE Multi-conference on systems and control*, St Petersburgh, Russia, 2009.

[5] L. Fesquet and H. Zakaria. Controlling energy and process variability in system-on-chips: needs for control theory. In *3rd IEEE Multi-conference on Systems and Control (MSC/CCA 2009)*, Saint Petersburg, Russia, July 2009.

[6] M. Krohn, E. Kohler, and M. F. Kaashoek. Events can make sense. In *USENIX Annual Technical Conference*, Santa Clara, CA, USA, June 2007. USENIX Association.

[7] H. Schwarz, D. Marpe, and T. Wiegand. Overview of the scalable video coding extension of the H.264/AVC standard. *IEEE Trans. on circuits and systems for video technology*, 17(9), 2007.

[8] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the H.264/AVC video coding standard. *IEEE Trans. on circuits and systems for video technology*, 13(7), 2003.

[9] C. C. Wurst, L. Steffens, W. F. Verhaegh, R. J. Bril, and C. Hentschel. Qos control strategies for high-quality video processing. *Real Time Systems*, 30(1), 2005.

[10] N. Zeldovich, A. Yip, F. Dabek, R. Morris, D. Mazières, and M. F. Kaashoek. Multiprocessor support for event-driven programs. In *USENIX Annual Technical Conference*, San Antonio, TX, USA, June 2003. USENIX Association.