Fine Grain QoS Control for Multimedia Application Software

Jacques Combaz[1,2], Jean-Claude Fernandez[1], Thierry Lepley[2], Joseph Sifakis[1]
[1]Verimag, Centre Equation - 2 avenue de Vignate F38610 Gières, France
[2]STMicroelectronics Central R&D 850, rue Jean Monnet 38921 Crolles Cedex, France

Abstract

We propose a method for fine grain QoS control of dataflow applications. We assume that the application software is described as the composition of actions (C-functions) with quality level parameters. The method allows to compute a QoS controller from this description, and average execution times, worst case execution times and deadlines for its actions. The controller computes dynamically feasible schedules and quality assignments for their actions. Furthermore, the control policy ensures optimal time budget utilization. A prototype tool implementing the method is shown as well as experimental results for a non trivial example. The results show the interest of fine grain QoS control for video encoders.

1. Introduction

Cost-effective development of embedded software should be based on the extensive reuse of generic software components. It should also be supported by tools and methodologies for guaranteeing given functional and extra-functional properties. In particular for multimedia embedded software the fast evolution of market needs, user requirements and platforms requires reliable adaptation of standard features at minimal costs. Currently, adaptation of available application software to target platforms and needs is too costly. To meet given QoS requirements a significant amount of experimentation is needed on virtual or real prototypes involving fine tuning of parameters of the components of the application software. After tuning, the behavior of application software can be modified only by changing user-defined input parameters. Thus, adaptability is coarse grain as it can be achieved only by modifying global parameters. Furthermore, some delay is necessary for adaptation due to limited controllability of the application software over the underlying execution system.

In this paper, we study a method for fine grain QoS control of multimedia applications meeting both soft and hard real-time requirements. For these applications uncertainty about execution times, makes necessary the use of control (dynamic scheduling) techniques [2],[10],[8]. The behavior of the controlled system is adapted by adequately choosing the values of quality levels for its actions. The objective of the control policy is both to respect deadlines for actions and to make optimal use of the time budget so as to reach best quality. The considered method allows to generate from an application software a controlled application software that meets given QoS requirements, as follows.

• The initial application software performs cyclically input/output transformations of data streams. It is described by a precedence graph modeling dependency between actions (C-functions) and from which all the possible execution sequences can be extracted. Its execution during a cycle can be controlled by choosing *quality levels* which are parameters of the actions. We assume that the execution times of actions are increasing with quality.

• We consider single threaded implementations of the application software on a platform for which it is possible by using timing analysis and profiling techniques, to compute estimates of worst-case execution times and average execution times of actions for the different levels of quality. Action execution is assumed to be atomic (non interruptible). A compiler is used to generate from the initial application software, for given QoS requirements and execution times, the controlled software. QoS requirements are deadlines on the termination of actions since the beginning of a cycle.

The controlled software can be considered as the composition of the initial application software with a *controller* (see figure 1).

• The controller monitors the progress of the computation in a cycle and chooses the next action to run and its quality level. It uses the model of the application software as well as knowledge for each action of its deadline and average worst case execution times.

• At some state of a cycle, the controller chooses quality levels guided by two kinds of constraints. Safety constraints ensuring that no deadline is missed during the cycle. Optimality constraints ensuring optimal time budget utilization, that is, the available time for completing an action is used as much as possible to achieve the best quality (without violating safety).

Our method significantly differs from existing ones for QoS control and adaptive scheduling. The main difference is fine grain control of the execution. Existing control techniques act at higher level e.g. at the beginning of a cycle, and their reactivity is slow. They do not require any deep knowledge of the data-flow structure of the application software. Our method consists in controlling execution during a cycle; the controlled software is produced by compilation (automatic code instrumentation).

Another important difference is that fine granularity allows combination of optimality and safety of the produced schedules. Most control techniques focus on optimality criteria and are adequate only for soft real-time. The integration of safety criteria is useful in applications where quality should remain above some minimal level [6],[3] or hard deadlines must be respected e.g. communications of cellular phones.

Buttazzo et al. propose the elastic tasks model [5], but there approach is based on worst case execution times. Another common and simple way to treat CPU overload is to skip an instance of a task [7]. Lu et al. [8] propose a feedback scheduling based on PID controllers, but deadline misses remain possible. Steffens et al. [10],[9] minimize deadline misses of an MPEG decoder by applying Markov decision process and reinforcement learning techniques, combined with structural load analysis.

The paper is organized as follows. Section 2 presents the method including a technical definition of the studied problem, an abstract control algorithm and its quality management policy. Section 3 reports on experimental results. A prototype tool is presented for generating controlled application as well as a non trivial example.

2. QoS Control

2.1. The Problem to be Solved

We need the following definitions about real-time systems and their scheduling.

Definition 2.1 *A* **real-time system** *is modeled by:*
• *a model of its application software, partial order on the vocabulary of its* actions A*, represented by its* precedence graph $G = (A, \rightarrow)$*, where* $\rightarrow \subseteq A \times A$*. We write* $a \rightarrow a'$ *for* $(a, a') \in \rightarrow$*.*
• $C : A \rightarrow \mathbb{R}^+ \cup \{+\infty\}$*, a function which associates with actions a their* execution time $C(a)$.
• $D : A \rightarrow \mathbb{R}^+ \cup \{+\infty\}$*, a function which associates with actions a their* absolute deadline $D(a)$.

An action a' can start only if the execution of all its predecessors a, $a \rightarrow a'$, is completed. Execution times of actions depend on the speed of the execution platform. Deadlines are user requirements that must be met by execution sequences formally defined below.

An *execution sequence* of a precedence graph G is a sequence of distinct actions $\alpha = \alpha(1) \ldots \alpha(n)$ such that the order induced by α is compatible with the precedence relation of G, that is $\alpha(i) \rightarrow \alpha(j) \;\Rightarrow\; i \leq j$, and for any prefix of α the set of the actions occurring in the prefix contains all its predecessors by the precedence relation.

For a given execution sequence α of G, an integer $i, 0 \leq i \leq |\alpha|$, defines a (control) *location*. If σ is a sequence over the non negative reals $\mathbb{R}^+$, then $\widehat{\sigma}$ denotes the sequence of length $|\sigma|$ where the i-th element is the sum of all the elements of rank $j \leq i$: $\widehat{\sigma}(i) = \sum_{j=1}^{i} \sigma(j)$. We write $\mathbf{min}(\sigma)$ to denote the minimum of the elements of σ.

We extend the functions C and D to execution sequences. If α is an execution sequence of length n, then $C(\alpha) = C(\alpha(1)), \ldots, C(\alpha(n))$ is the sequence of the execution times of the elements of α. Similarly, $D(\alpha)$ is the sequence of the deadlines of the elements of α.

Definition 2.2 *Let G be a precedence graph, C an execution time function and D a deadline function. A* **schedule** *of G is an execution sequence* α *where occur all the actions of A. A schedule* α *is* feasible *with respect to C and D if:*

$$\mathbf{min}\left(D(\alpha) - \widehat{C(\alpha)}\right) \geq 0$$

Notice that the above definition means that the schedule respects the deadlines for the given execution times.

For systems with known execution times, feasible schedules can be computed statically e.g. by computing EDF schedules [4]. When execution times are not precisely known, static computation of feasible schedules requires the use of worst case execution times. This may lead to solutions that are far from being optimal, especially in the case where uncertainty about execution times is high, that is the difference between average and worst case execution times can be significant.

We present a method for computing a controller which composed with the application software, ensures satisfaction of deadlines as well as optimal use of the available time budget by adapting the quality levels of actions. The controller uses an abstract model of the real-time system to be implemented.

Definition 2.3 *A* **parameterized** *real-time system is defined by:*

• *a precedence graph G,*
• *a finite set $Q \neq \emptyset$ of integers, called* quality levels
• *for each quality level $q \in Q$, C_q^{av} and C_q^{wc}, such that $C_q^{av} \leq C_q^{wc}$, are execution time functions which are not decreasing in q. They give for any action a its average execution time $C_q^{av}(a)$ and its worst case execution time $C_q^{wc}(a)$*
• *for each quality level $q \in Q$, D_q is a deadline function associating for any action a, its deadline $D_q(a)$.*

For a family of time functions $\{X_q\}_{q \in Q}$ and $\theta : A \to Q$ a quality assignment function, X_θ is a time function such that $X_\theta(a) = X_{\theta(a)}(a)$. Let $q_{min} \in Q$ be $\mathbf{min}(Q)$.

The problem to be solved can be formalized as follows.
Problem: Consider a parameterized real-time system such that the set of the feasible schedules with respect to $C_{q_{min}}^{wc}$ and $D_{q_{min}}$ is non empty. Find a controller which computes schedules α and quality assignment θ such that for any execution time function C, $C \leq C_\theta^{wc}$:

- α is a feasible schedule with respect to C and D_θ
- θ is an optimal quality assignment (in a sense to be defined later).

Notice that in the above definition, C is an arbitrary execution time function that describes the unpredictable execution times of the controlled system. We call C *actual* execution time function. The requirement $C \leq C_\theta^{wc}$ is essential for safe control. It says that execution times of the controlled system cannot be longer than the worst case execution times (for the same quality).

2.2. Abstract Control Algorithm

The controller computes incrementally a schedule α^n and a quality assignment θ^n for $n = |A|$. This is carried out by computing successively pairs (α^i, θ^i) for $i = 1, \ldots, |A|$ corresponding to the i-th computation step. Clearly, two successive pairs (α^i, θ^i) and $(\alpha^{i+1}, \theta^{i+1})$ must be compatible in the sense that the prefixes of length i of α^i and α^{i+1} are the same, and the restrictions of θ^i and θ^{i+1} on their elements, agree.

Figure 1 illustrates the principle of computation of a new pair $(\alpha^{i+1}, \theta^{i+1}) = (\alpha', \theta')$ from the current pair $(\alpha^i, \theta^i) = (\alpha, \theta)$ and the actual time $\widehat{C(\alpha)}(i) = t$. By construction, at step i, all the elements of α^i after the i-th position have a constant quality assignment. The controller is decomposed into two cooperating components, a *Scheduler* and a *Quality Manager*. For a schedule α, we denote by $\alpha[i, j]$ the sequence $\alpha(i) \ldots \alpha(j)$, and define the following predicates:
• *Qual_Const*$^{av}(\alpha, \theta, t, i) =$
$$t \leq \mathbf{min}\Big(D_\theta(\alpha[i+1, n]) - C_\theta^{av}(\widehat{\alpha[i+1, n]})\Big)$$
• *Qual_Const*$^{wc}(\alpha, \theta, t, i) =$
$$t \leq \mathbf{min}\Big(D_{\theta'}(\alpha[i+1, n]) - C_{\theta'}^{wc}(\widehat{\alpha[i+1, n]})\Big)$$ where $\theta'(\alpha(j)) = q_{min}$, for $j > i+1$, $\theta'(\alpha(j)) = \theta(\alpha(j))$ otherwise.
• *Qual_Const* $=$ *Qual_Const*$^{av} \wedge$ *Qual_Const*^{wc}.

Notice that *Qual_Const*^{av} (resp. *Qual_Const*^{wc}) means feasibility for α with respect to C' (resp. C'') and D_θ (resp. $D_{\theta'}$). The functions C' and C'' are such that $C' = C'' = C$ for $\alpha(1), \ldots \alpha(i)$, and $C' = C_\theta^{av}$, $C'' = C_{\theta'}^{wc}$ otherwise.

The *Scheduler* and the *Quality Manager* cooperate in the following manner:

1. For different quality values q the Quality Manager computes $\theta_q := \theta \rhd_i q$, quality assignments which agree with the current quality assignment θ for the first i elements of α and give q for all the others.
2. For each θ_q, the Scheduler computes, by using some optimal scheduling algorithm, e.g EDF, a schedule α_q which has the same prefix of length i with α.
3. The Quality Manager computes the quality level q_M which is maximal and meets the quality constraint *Qual_Const*$(\alpha_q, \theta_q, t, i)$ which characterizes all the acceptable schedules and quality assignments.

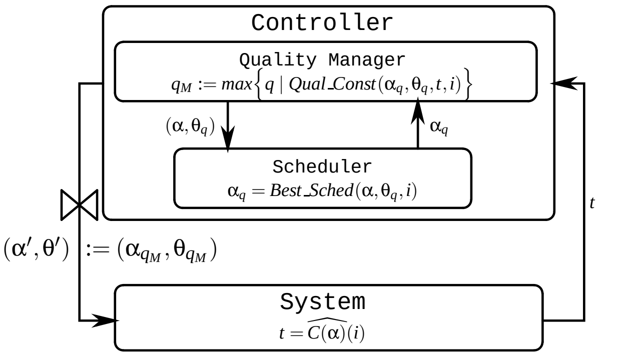
Controller
Quality Manager
$q_M := max\{q \mid Qual\_Const(\alpha_q, \theta_q, t, i)\}$
(α, θ_q)
α_q
Scheduler
$\alpha_q = Best\_Sched(\alpha, \theta_q, i)$
t
$(\alpha', \theta') := (\alpha_{q_M}, \theta_{q_M})$
System
$t = \widehat{C(\alpha)}(i)$

Figure 1.

The controller executes the following abstract algorithm. We use a variable i to represent the current computation step and variables α and θ such that, at step i, $\alpha = \alpha^i$ and $\theta = \theta^i$. Furthermore, we use sets of variables $\{\theta_q\}_{q \in Q}$ and $\{\alpha_q\}_{q \in Q}$ such that at step i their values are $\theta_q = \theta_q^i$ and $\alpha_q = \alpha_q^i$.

```
$i := 0$
while $i < |A|$ do
    for $q \in Q$ do $\theta_q := \theta \rhd_i q$
    for $q \in Q$ do $\alpha_q := Best\_Sched(\alpha, \theta_q, i)$
    $q_M = \mathbf{max}\ \{q \mid Qual\_Const(\alpha_q, \theta_q, t, i)\}$
    $(\alpha, \theta) := (\alpha_{q_M}, \theta_{q_M})$
    $i := i + 1$
end while
```

We proved the following proposition:

Proposition 2.1 *For any schedule* α *and quality assignment* θ *computed by the algorithm,*

- **safety:** α *is a feasible schedule with respect to* C *and* D_θ
- **optimality:** *the time budget utilization* $\widehat{C(\alpha)}(n)/D_\theta(\alpha)(n)$ *is maximized.*

3. Experimental Results

We applied these results to an MPEG 4 encoder provided by STMicroelectronics and written in C (more than 7000 loc). The encoder treats frames cyclically. Each frame is split into N macroblocks of 256 pixels. It can be considered as the iteration N times of a body whose precedence graph is given in figure 2. The overall architecture is shown in figure 3. It uses input and output buffers of the same size K, to cope with changes of load and avoid as much as possible frame skips. These may happen when the input buffer is full.

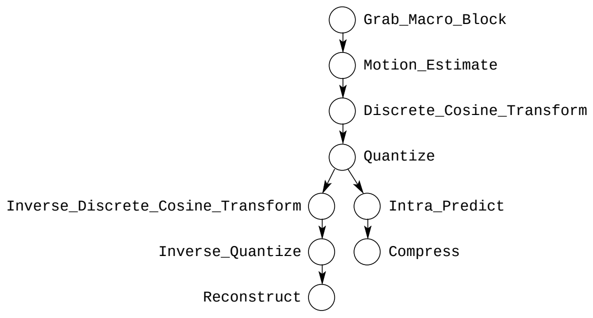

Figure 2.

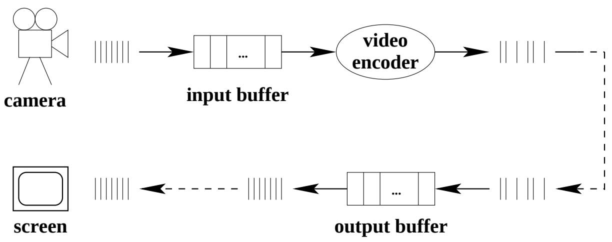

Figure 3.

We developed a prototype tool (figure 4) that allows the generation of controlled application software when the order between the deadlines is independent of the quality. The inputs of the tool are
• the precedence graph G corresponding to the treatment of a macroblock and its iteration parameter N,
• tables describing the functions C^{av} and C^{wc} for the actions of G,
• the order relation between the deadlines.

From these inputs the tool computes
• C code corresponding to an EDF schedule α.
• tables containing pre-computed values used by the controller for the computation of $Qual\_Const^{av}$ and $Qual\_Const^{wc}$.

A compiler is used to link the following items and generate the controlled application software from
• the schedule and the tables generated by the tool
• application code for the actions of the schedule
• a generic controller mainly consisting of a quality manager.

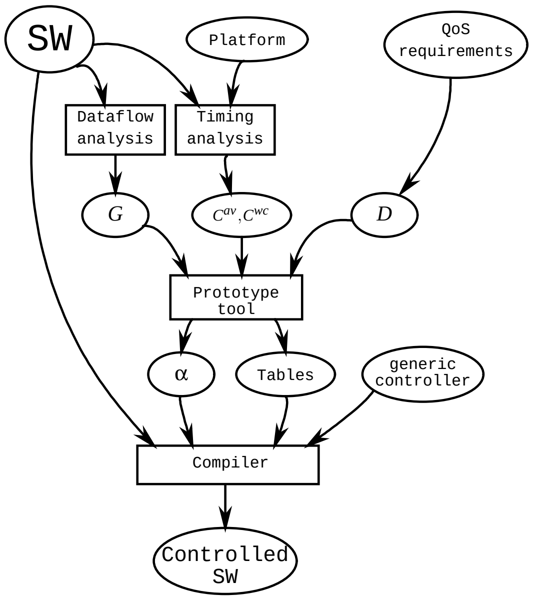

Figure 4.

The overhead due to the instrumentation of the application software in the size of the compiled code is of the order of 2% for the considered benchmarks. During execution, the corresponding overhead in memory allocation is not more than 1%. Finally, the overhead in runtime is estimated less than 1.5% of the overall execution time. For all these estimates we assume that the application software runs on a single processor without OS and that it is possible to read a register counting the number of cycles elapsed.

For the considered example, the execution times of the action `Motion_Estimate` depend on the quality level as specified in figure 5. The execution times of all the other actions are independent of the quality levels, and are given in same figure 5.

We provide experimental results for a platform consist-

Motion_Estimate		
Quality	**Average**	**Worst case**
0	215	1000
1	30000	100000
2	50000	200000
3	95000	350000
4	110000	500000
5	120000	1200000
6	150000	1200000
7	200000	1500000

Action	Average	Worst case
Grab_Macro_Block	12000	24000
Discrete_Cosine_Transform	16000	16000
Quantize	6000	13000
Intra_Predict	4000	4000
Compress	5000	50000
Inverse_Quantize	4000	5000
Inverse_Discrete_Cosine_Transform	20000	50000
Reconstruct	10000	13000

Figure 5.

ing of a single XiRisc processor [1] running at 8 GHz. This corresponds to the computing power of embedded parallel architectures used for video encoding. The platform is simulated by using the eliXim tool of STMicroelectronics. Time unit is a CPU cycle.

We consider a benchmark of 582 frames, consisting of 9 sequences produced by a camera every $P = 320$ *Mcycle* (i.e. constant framerate of 25 $frame/s$). The target bitrate is set to 1.1 $Mbit/s$.

The buffers of size K allow a maximal latency of $P \cdot K$. The time budget allocated to the encoder for the treatment of a frame depends on the buffer occupancy, and is in average P. As our method guarantees safety, we can take $K = 1$ for the controlled encoder without deadline miss.

We measure PSNR between the input frames and output frames, as well as the utilization of the average time budget which is the ratio between the time for encoding a frame and P, as a function of the number of treated frames. PSNR characterizes single frame quality and is used to measure the effect on video quality of the encoding process. We compare the controlled encoder generated by our prototype and the same encoder for constant quality level (this corresponds to standard industrial practice).

Time budget utilization is shown in figures 6 and 7, for controlled quality, constant quality $q = 3$ and $K = 1$, and constant quality $q = 4$ and $K = 2$. Notice the presence of two kinds of jumps: eight jumps corresponding to changes of video sequences (encoding of I-frames); two bursts of jumps corresponding to frame skips due to buffer overflow occuring for constant quality only.

PSNR for the same test cases are given in figures 8 and 9. Notice again the two kinds of jumps due to changes of video sequences and frame skips. When a frame is skipped, the immediately previous frame is displayed by the decoder, and the comparison to the input frame gives a low PSNR

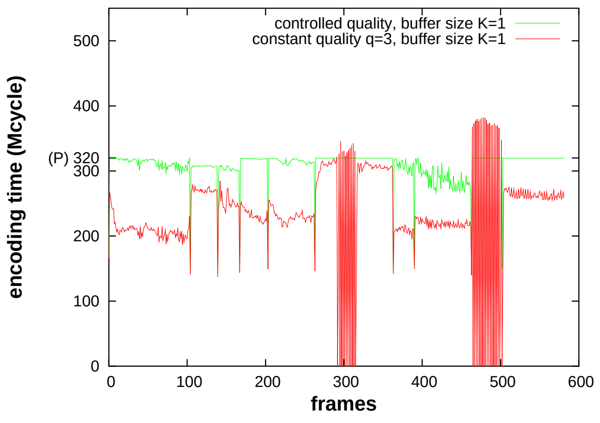

Figure 6. Time budget utilization.

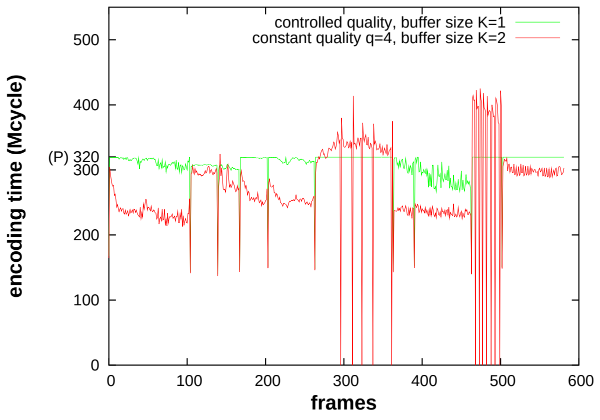

Figure 7. Time budget utilization.

value (e.g lower than 25).
In figure 8, PSNR is higher for controlled quality than for constant quality $q = 3$, except for regions where frames are skipped. For these regions, the bits corresponding to skipped frames are used to achieve better quality. Although the PSNR is higher in these regions for constant quality, the video quality is affected as the frame rate is divided by two.
In figure 9, using buffers of size $K = 2$, allows to activate constant quality 4 with a reasonable amount of skipped frames. As in the figure 8, the PSNR of the controlled application is higher except in regions where frames are skipped.

Experimental results show that for constant quality levels load fluctuation can lead to poor video quality in absence of sufficiently large buffers. Poor video quality means low PSNR or frame skips (or both). For controlled quality levels, there are no frame skips. Thus, overloads result in low PSNR. Furthermore, using buffers may not completely eliminate frame skips, implies additional cost and increases latency. The comparison between constant and controlled

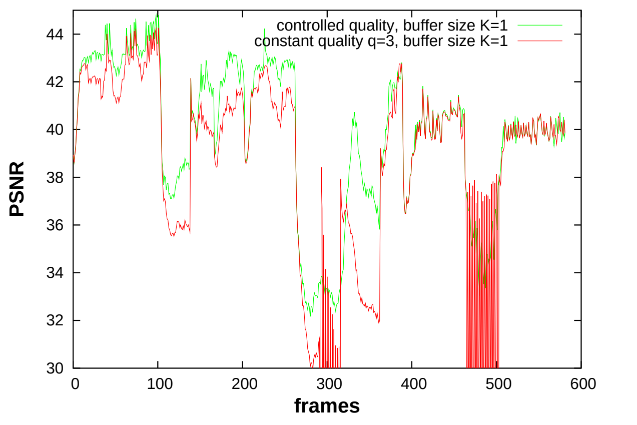

Figure 8. PSNR between input and output.

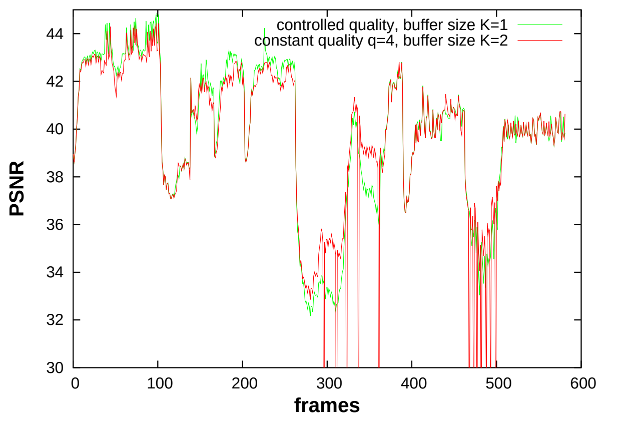

Figure 9. PSNR between input and output.

quality shows that for controlled quality we get better video quality even for buffer size 2. Controlled quality completely avoids frame skips; overloads lead to smooth reduction of PSNR.

4. Conclusion

The presented method uses fine grain control for hard and soft real-time requirements. It overcomes some of the limitations of hard real-time approaches where strict respect of deadlines implies poor time budget utilization. This is possible because of the use of fine grain control, which allows adaptation to changing load during a cycle instead of using a priori known global execution time estimates. Notice also that our method can be applied to systems with hard and soft deadlines. For soft deadlines, the Quality Manager applies only the average quality constraint.

The method relies on a solid theoretical base. We proved that the control policy is safe under some reasonable assumptions about the controlled system — actual execution times are less than worst case execution times. We also proved optimal budget utilization. Furthermore, we studied specific conditions guaranteeing smoothness in terms of variations of quality levels chosen by the controller.

Experimental results confirm the interest of the method and its low overhead. Given their importance, we actively work in several directions to improve the prototype tool: compositional generation of EDF schedules for iterative programs, specialization of the *Best_Sched* function, application of learning techniques for better estimation of the average execution times as well as heuristics for optimal control.

In parallel, we explore theoretical foundations of this control technique and its interrelations with existing ones. The most important difference is fine grain QoS control, not at system or task level, but at software execution level. This requires a fine analysis of the application software to extract its model as well as the use of compilation techniques to generate the controlled application. A crucial issue in this low level control is efficiency of the implementation to avoid overhead and probe effects due to instrumentation.

References

[1] http://xirisc.deis.unibo.it/.

[2] K.-E. Arzen, B. Bernhardsson, J. Eker, A. Cervin, K. Nilsson, P. Persson, and L. Sha. Integrated control and scheduling. Technical report.

[3] R. J. Bril, M. Gabrani, C. Hentschel, G. C. van Loo, and E. F. M. Steffens. Qos for consumer terminals and its support for product families. In *Proceedings of the International Conference on Media Futures*, 2001.

[4] G. C. Buttazzo. *Hard Real-Time Computing Systems*. Kluwer Academic, 2000.

[5] G. C. Buttazzo, G. Lipari, and L. Abeni. Elastic task model for adaptive rate control. In *RTSS*, pages 286–295, 1998.

[6] D. Isovic, G. Fohler, and L. Steffens. Timing constraints of mpeg-2 decoding for high quality video: misconceptions and realistic assumptions.

[7] G. Koren and D. Shasha. Skip-over: Algorithms and complexity for overloaded systems that allow skips. Technical Report TR1996-715, , 1996.

[8] C. Lu, J. Stankovic, G. Tao, and S. Son. Feedback control real-time scheduling: Framework, modeling and algorithm. *special issue of RT Systems Journal on Control-Theoric Approach To Real-TIme Computing*, 23(1/2):85–88, 2002.

[9] L. Papalau, C. M. O. Pérez, and L. Steffens. In S. Goddard, editor, *Work-In-Progress Session of the 16th Euromicro Conference on Real-Time Systems*, pages 33–36, 2004.

[10] C. C. Wüst, L. Steffens, R. J. Bril, and W. F. Verhaegh. Qos control strategies for high-quality video processing. In *Euromicro Conference on Real-Time Systems*, pages 3–12. IEEE, 2004.