49th IEEE Conference on Decision and Control
December 15-17, 2010
Hilton Atlanta Hotel, Atlanta, GA, USA

Controlling software applications via resource allocation within the Heartbeats framework

Martina Maggio[1,2], Henry Hoffmann[2], Marco D. Santambrogio[1,2], Anant Agarwal[2], Alberto Leva[1]
[1]Dipartimento di Elettronica e Informazione
Politecnico di Milano, Italy, 20133
{maggio, santambr, leva}@elet.polimi.it
[2]Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology, Cambridge, MA 02139
{mmaggio, hank, santa, agarwal}@csail.mit.edu

Abstract— **A formalism was recently introduced to instrument, monitor and control computer applications based on the rate of *heartbeats* they emit, thereby quantitatively signaling their progress toward goals. To date, the idea was however used essentially in an heuristic manner. This work first shows that a very simple dynamic heartbeat rate model can be devised, an that said model allows to address the corresponding control problems in a methodologically grounded way. A general solution is then devised, that can be realized through different actuation mechanisms, depending on which type of resource—CPU, memory, bandwidth, etc.—is constraining the application performance in the particular situation at hand. Experiments prove the efficacy of the proposed extension to the heartbeats framework, both with applications that fit the proposed model and with more complex test cases, for which said model is just a coarse approximation.**

I. Introduction

Modern computing systems, from multicores to clouds, present a number of challenges to software developers. In addition to performance and correctness issues, modern systems must be concerned with power and energy efficiency as well as reliability and predictability, even when the execution environment cannot be characterized a priori. One approach to addressing these challenges is to build computing systems that observe their execution and adjust their behavior based on feedback from said observations. Recently, the control community has been active in studying such computing systems [1], [2]. However most of the literature is concerned with the analysis of existing systems, whose complexity and inherently heterogeneous nature requires equally complex modeling paradigms - and often identification mechanisms - to be exploited [3].

In fact, most adaptive computing systems problems can be formulated as control ones in a direct manner. For example, consider the problem of allocating processor cores in a modern multicore processor. A possible strategy is to have the system allocate the minimum number of cores required to meet application goals. Unused cores can be allocated to another application or possibly idled to save power. Such a system could greatly reduce the programmer's burden by ensuring that the application always uses the minimum number of cores to reach its performance goal. To complete this formulation, however, there must be a generalized interface for an application to make both its goals and current performance known to the resource allocator. Such a general interface would allow any possible controller to work on the active applications.

It has recently been observed that feedback control allows the reformulation of many computing system management problems in an effective way, leading to simpler solutions than the ones currently adopted, provided some *instrumentation* is introduced. Curiously enough, this observation has been made in both the control and the computer science communities: examples are respectively [4] and the Application Heartbeats framework [5], used and extended in this work.

The software framework called Application Heartbeats, has been recently introduced to make it possible to monitor generic applications. Using this interface, applications express their performance by registering a "heartbeat" at important intervals. Additionally, the interface allows applications to express their performance goals in terms of a desired heart rate or a desired latency between specially tagged heartbeats. Thus, the Application Heartbeats interface allows any application to export a feedback mechanism upon which a control system can be devised. For example, a video encoder can be written to produce a heartbeat with every frame of video. Furthermore, this encoder can express its target performance as a desired rate of frames per second. With this information, the controller (i.e., the resource allocator) described above could minimize the amount of computational resources required to meet the encoder's performance goals.

This paper develops and analyzes a control system for allocating resources in a multicore processor using Application Heartbeats to provide the performance target and the feedback mechanism. To begin, related work is discussed in Section II. Next, a model of the system to be controlled is presented in Section III. This model is simple, yet still captures the control-relevant dynamics. It also incorporates disturbances, to reasonably account for the effects of a non-structured external environment or of unmodeled dynamics with a minimum additional effort. After discussing this model, the control strategy is described in Section IV. The key property of this controller is that the control signal computation is completely disjoint from the actuation policy. In this work, the selected actuation policy is the assignment of cores to the application; however, the controller is general enough to work with any actuation policy which can modify the speed of the application. After presenting the control strategy, some experiments demonstrate the use of this strategy to control a real computing system. Specifically, these experiments (in Section V) illustrate how the controller is able to allocate the minimum number of cores required to maintain a video encoder's target performance while encoding high-definition video. The paper concludes in Section VI.

II. Motivation and Related work

Control theory is emerging as an approach for the *design* of self-managed computing systems. Karamanolis *et al.* stress the need for a rigorous approach in the design of dynamically controlled systems and many computing management problems can be formulated such that standard controllers can be applied [6]. This formulation requires the necessary tunable parameters (actuators) and the appropriate feedback metrics (sensors and measurements).

Several examples show the advantages of such a formulation. Hollot has shown how the congestion window mechanism in the TCP protocol can be interpreted as a control system [7]. Scheduling policies were addressed as well, and [8] points out the important distinction between open-loop and closed-loop algorithms. Encoding and/or decoding videos maintaining adequate service levels while keeping the energy consumption as low as possible is a open problem. Bang *et al.* refer to video decoding and shows some comparisons between existing techniques before discussing a new approach based on workload estimation via Kalman filters [9]. In [10] the same problem is faced for parallel data-flow applications. The application is described as a pipeline of stages and the goal is to find the correct speed configuration for each element of the pipeline to reach the goal minimizing the overall energy consumption. In [11] a control structure is proposed, however not yet realized, in which a feedback loop allows the translation of the service level agreement objectives into set-points for the response time of the servers and tracking performance is traded-off with energy savings. Varma *et al.* present a modification of the PID control algorithm to synthesize a nqPID (not quite PID) control system to chose the CPU frequency of a computing unit [12]. In [13] a system that runs multiple receding horizon controllers on the same multi core machine is considered. The number of processors assigned to each subsystem is changed as well as the execution horizons of each controller, casting the proposed problem into an optimization one. This work addresses the problem for a known environment, where a predefined number of processes compete for the same resource. These processes are however very well characterized and their behavior is a priori known.

None of the mentioned contributions, however, relies on a general software framework that allows the goal definition and the progress measurement in a uniform way. In this paper the Heartbeats framework is used as such a general framework and a single application is considered. The main contribution of this manuscript is a formulation of heart beat rate based problems in terms of feedback control ones through a prior definition of a model of the application rate dynamics, and subsequently a simple but methodologically grounded control design. Note that computational lightness is a must in this context, so that for any possible solution simplicity is a merit.

III. The model of the controlled system

As pointed out and discussed in [4], the successful formulation of computing system control problems relies on a clear definition of the system to be controlled prior to the control design—a less trivial task than it may appear.

The aim of this section is to carry out the above task in the context of this work, i.e., to propose a simple yet promising model of a generic software application, which is conceived to be suitable for control purposes. First, the model, which makes use of the heart beat abstraction, is presented. Second, its accuracy is discussed, showing the advantages and the limitations of the outlined approach.

A. Hypotheses and formulation

The model assumes that applications have been instrumented with a feedback-oriented mechanism, in this case Application Heartbeats. Thus, an application issues a series of heartbeats at regular intervals. In addition, the application makes its target performance, or heart rate measured in heartbeats per second, known to the rest of the system.

Consider a single heartbeat-enabled application executed in a computing system with N_c processor cores. Denote by k the index counting the heartbeat signaling operations. Also, let $hb_t(k)$ be the time elapsed between the $(k-1)$-th and the k-th measurement. Define by w_σ the workload of the instrumented application—the workload is defined to be the average expected time between two subsequent heartbeats when the application is executed with the minimum available amount of needed resources (i.e., with a single core). Assume that the workload does not depend on the application's progress, so that it can be considered known and constant (relaxations will be addressed in the future). Under the above assumptions, a model for the system under control (the instrumented application) takes the form

$$hb_t(k) = w_\sigma \, u(k-1) + \delta w(k-1) \qquad (1)$$

where δw is an exogenous disturbance and it is assumed that the actual time $hb_t(k)$ does not depend on the previous execution times $\{hb_t(k-1), hb_t(k-2)\ldots\}$. This is a realistic assumption for a lot of applications, as long as they are carefully instrumented. However, it does not always hold. Think for example at the aforementioned video encoder that produces a heartbeat with every frame of video. The frames can be divided into three different categories: I, P and B frames [14]. I frames are difficult to compress and do not require other video frames to decode after compression. P frames can use data from previous frames to decompress and therefore their content can be represented with less data and encoded differently. B frames can use both previous and forward frames for data reference to get the highest amount of data compression. The use of these different types of frames means that the workload can vary as a function of frame-type and in some implementations the time to encode a single frame can in principle depend on the time taken in the previous ones' encoding.

In control problems tackled with model (1), the phenomenon just sketched appears as unmodeled dynamics. Of course one could extend the model to handle the different frame-types, but the new model would become application-specific. Therefore, although model extensions can (and will) be considered, this work aims at proving that good results can be obtained while keeping the model as simple (and general) as possible.

As a side remark notice that the same model could be written with the heart rate in place of the time between subsequent heart beats with no influence on the results. Moreover, the transfer function from the control signal u to the time hb_t is

$$\frac{HB_t(z)}{U(z)} = \frac{w_\sigma}{z}. \tag{2}$$

Such a control signal $u(k)$ represents the fraction of time the application should consume with respect to the workload, and is thus bounded in the interval $(0,1]$. Having $u(k)=1$ is equivalent to giving the application the least amount of resources, and limiting its execution time to the workload. Suppose that $u(k) \geq \varepsilon$; $u(k)=\varepsilon$ means allowing the application to run with all the available resources in order to obtain the maximum speed-up. Notice that the control signal choice is completely general with respect to the actuation policy, so the same control signal will work with different actuation policies. For example, this control signal can be used with an actuation policy which allocates cores to speed up compute-bound applications or with an actuation policy which assigns DRAM bandwidth to speed up memory-bound applications.

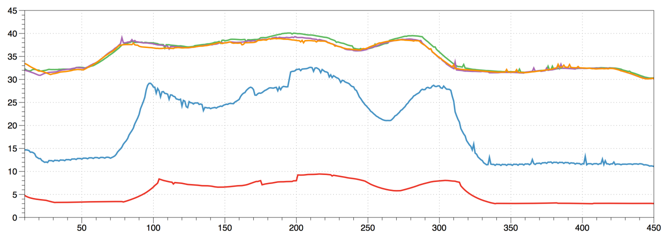

Fig. 1. Heartbeat rate for x264 application launched with 1 (red), 4 (light blue), 16 (green), 24 (purple) and 32 (orange) threads when no control is applied. In abscissae the number of heart beat for the encoding of 512 frames, an heartbeat is emitted every time the application complete the encoding of a single frame.

As anticipated, the proposed model does not always fit the application. Sometimes the application execution is marked by different phases with the workload varying among these phases [15]. In order to model this behavior, the workload can be seen as an exogenous switching signal [16] that changes when the application enters a different phase. Figure 1 reports an example of a instrumented version of x264 [17], running with an input that exposes phases in its behavior. The figure depicts the heart rate, i.e., the number of heartbeats per second, when the application has to encode 512 different frames[1] and is launched respectively with different number of threads on a 16 core machine. In the following however, we will make the simple assumption of a constant workload, while future studies will be devoted to the analysis of the system as a switching one. If the workload profile is known a priori, the devised controller is still valid, as long as the workload value is updated.

B. Some words on accuracy

The model used this work is very simple, yet it captures all the necessary dynamics to properly control the system and the presence of disturbances accounts for workload variations that are not a priori predictable. However, the model requires mapping $u(k)$ into an effective control action. This mapping is conceptually part of the control mechanism, but can invalidate the model if not properly considered (see again the discussion in [4]).

Finding a good actuation policy is a complex problem [6]. In order to take advantage of the modeling analysis done so far, it is necessary to describe the relationship between the control variable $u(k)$ and the effective actuation (i.e., the number of cores the application can use, a frequency scaling mechanism, a scheduling policy, a memory allocation algorithm, etc.). If such a relationship is not valid for the current application, the control system will not behave as expected.

For example, if we are controlling an application by changing the number of cores it can use, the proposed model does not take into account the average, expected parallelism of the algorithm realized by the application in the k-th sampling period. This value can be introduced but will lead to a more complex formulation. Generally speaking, the average parallelism is defined as the average number of busy processor cores during the considered period when an unlimited number of cores are available [19]. Assigning more cores than this value would not be beneficial. If available, average parallelism for an application can be used to produce a more accurate control system, but will result in a more complicated model.

Generally, problems arise when the control actuator allocates a particular resource, but changing that resource does not influence the application behavior. For example, think of an application that is waiting to retrieve data from an input device. No matter how many cores the controller assigns, the application will stall waiting for the input data. In such a scenario the model is not valid. This scenario can be modeled as a faulty actuator, which makes the system behave in open-loop, even though the controller is still active. A proper antiwindup mechanism can manage the recovery of such a situation (i.e., regain the set point quickly once the actuator

[1]The input video in this case is taken from the PARSEC benchmark suite [18].

becomes effective again) but there is apparently a strong need for control solutions that change actuation policy when the system is not approaching the target goal (and are simple enough for the addressed context, where many fault detection techniques would be too complex to apply).

For the purpose of this work, we limit the analysis to one control solution and show how the control system behaves in two scenarios. First a simple example illustrates the behavior of the control system when the application depends only on the controlled resource. A second example explores the case where this assumption does not hold.

IV. A SIMPLE CONTROL SCHEME

Once the process model has been written, a control scheme needs to be synthesized. We use the standard scheme of Figure 2. A deadbeat controller is developed, to assign the transfer function between the desired time between two subsequent heartbeats hb_t° and the effective one hb_t.

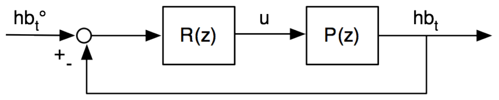

Fig. 2. The proposed control scheme.

The controller sets such a function to $1/z$, obtaining a pure delay. The regulator $R(z)$ is thus

$$R(z) = \frac{z}{(w_\sigma)(z-1)}. \quad (3)$$

Moreover, the set point hb_t° is generated inverting the desired heart rate, specified by the application through the Heartbeats framework.

As discussed above, the calculation of the control signal $u(k)$ has to be translated into a corresponding actuator value. A function that maps $u(k)$ into the corresponding actuator value has to be provided for every actuation mechanism. Although many actuators can be chosen, this work considers only the number of cores allocated to the instrumented application.

In the following, let $c(k)$ be the number of cores allocated to the instrumented application at time k. As a modeling parameter, it is assumed that the speedup of the application varies with the square root of the number of cores assigned to the software program. Thus,

$$u(k) = \frac{1}{\sqrt{c(k)}}, \quad c(k) = \frac{1}{u(k)^2} \quad (4)$$

and the control signal $u(k)$ is limited with an antiwindup mechanism in the range $[0.25, 1]$, i.e., the actuatot is limited to a maximum of 16 cores, which corresponds to the machine used in our experimental setup.

Obviously, in the proposed solution, only some values of $u(k)$ can be effectively mapped into the corresponding value for $c(k)$, as fractions of a core cannot be assigned. However, such a granularity results in a strong limitation of the control action. Therefore a time division output actuation scheme is adopted. For example, if in a given period 2.75 cores are required, the system can allocate three cores for the first 75% of that period and two for the remaining 25%. To achieve better performances, the period of such a scheme should depend on the estimated workload of the application and on the desired heart rate. The following section presents some examples.

V. EXPERIMENTAL RESULTS

In order to validate the proposed approach we report some tests, with two different software applications. First we test the system with an application (termed the "nominal application") that completely fits the proposed model, and then we report some case studies with the aforementioned video encoder x264 and different input data. The first test shows that when the model is correct the proposed control scheme works on the *real* device, including the control quantization, the time division output, and all the implementation-related overhead. The second set of tests shows that in a more complex case, where the proposed model is just an approximation, satisfactory results can be obtained anyway, although there is clear evidence of the need for future refinements.

A. Nominal application

Most applications require more than a single resource. However, there are interesting test cases where the software code is CPU-bounded. To reproduce this situation within the Heartbeat framework, we wrote a test application that behaves according to the proposed model (1). In such an application the workload variations are very small with respect to the model and the application scales almost exactly with the square root of the number of cores. A real software application was used instead of a simulation example, to validate the approach even in the presence of possible problems and delays due to the software implementation of the control scheme.

Figure 3 shows the results of the control action on this software application. As can be seen, the desired heart rate is reached in the minimum amount of time. The control action is performed every period, where such period is equal to the desired amount of time among 5 subsequent heart beats. As can be seen, the desired heart rate is 3.5 beats per second, and the period is $1428571\mu s$. The control system in this case easily lets the application meet its goals, with minimum resource consumption.

B. x264

As anticipated, the x264 video encoder is a complex application to control. It is not necessarily CPU bound during all parts of its execution, and the encoding time for a frame can vary. We are aware that more can be done by adding complexity to the model, but our experiments show that good results can be achieved if the involved parameters are correctly tuned even with the simple model proposed herein. In the following, two test cases are presented that refer to different input data. The first one has normal variations in the

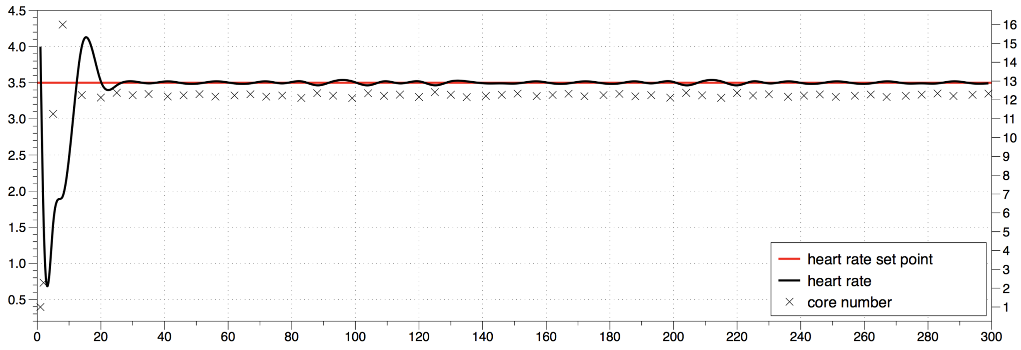

Fig. 3. Results with the nominal application in abscissæ the current heart beat, while the black curve is the heart rate, the points are the number of cores assigned before the time division output.

workload, while in the second case the application behaves according to the three distinct phases depicted in figure 1, and is thus more difficult to control.

Define w_s as the control window size, that is related to the control period. A control action is performed every T_s seconds, where such period is equal to the desired amount of time among w_s subsequent heart beats, being therefore $T_s = w_s / hb_t^{\circ}$. Typically, in a video encoding application the desired encoding speed is 30 frames per second. The instrumented application emits an heart beat every time a frame is completely encoded, so the desired heart rate 30 beats per second.

Figure 4 reports the first test results. Different experiments are conducted, with $w_s = 5, 15, 20$ and the corresponding periods equal to $w_s/30s$. In this test case, the estimation of the workload provided to the controller (1 second per heartbeat on a single core machine) is higher then the effective workload (an average value is 0.3 seconds per heartbeat when the application is executed with a single core). In the first plot, the measured heart rate is depicted in black, with different window sizes. The green curve shows the heart rate, when $w_s = 40$ and a correct estimation of the workload is provided. In the second plot, the corresponding control actions are shown. As can be seen, increasing the window size results in a more accurate but slower control action. A slower period results in heavier disturbance, while a larger period is preferable to limit the effects of the non-modeled dynamics. As we expect, $c(k)$ converges to the desired value in all cases.

The second test case is more complicated and Figure 5 shows the experimental result obtained when the workload estimation provided is 0.2 seconds per heartbeat, which is a reasonable value. Two different setup results are depicted, with $w_s = 20, 40$ and the corresponding periods equal to $w_s/30s$. As anticipated, in this second test case, the application does not completely fit the proposed model and exposes high variability. Moreover, such variability makes it useless to increase the window size in order to obtain a better result. This is probably due to the fact that in some of the application phases, the relationship between the actuation mechanism and the control signal does not hold anymore. However, the results are still valuable as a first exploration.

VI. Conclusion and future work

In this work the problem of controlling the resources assigned to a software application in order for this application to match its desired goal was considered, limiting the scope to the core allocation mechanism. The Heartbeats framework was exploited as a viable instrument, both for allowing the application to express its desires and to measure its performance in terms of how close the effective behavior is to the desired one.

A model of the system under control was discussed, highlighting its limitation and its advantages. The model can be further extended including other available information. A simple control solution was proposed, that relies on such a model. Simplicity and computational efficiency is a merit of the proposed research, given that every resource used to execute the controller is effectively subtracted from the application itself, in this case the computation time being the controlled resource. Nonetheless, more complex control schemes can be considered as well, considering for example the disturbance rejection as a primary objective.

Some experimental results for the devised solution were presented, showing that the controller behaves well for CPU bound applications—i.e., in general, when the available actuator is effective. It is worth noticing, from this point of view, that the proposed control action is in principle general, and different actuators can realize it. Future work will thus be devoted to analyzing such solutions, in which the system has different actuation mechanism and the controller can dynamically chose which is the preferred one, based on the application progresses.

Acknowledgment

We would like to thank the *Progetto Roberto Rocca* for its support.

References

[1] M. Shor, K. Li, J. Walpole, D. Steere, and C. Pu, "Application of control theory to modeling and analysis of computer systems," in *Proceedings of Japan-USA-Vietnam Workshop on Research and Education Systems*, HoChiMinh City, Vietnam, 2000.

[2] J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury, *Feedback Control of Computing Systems*. Wiley, 2004.

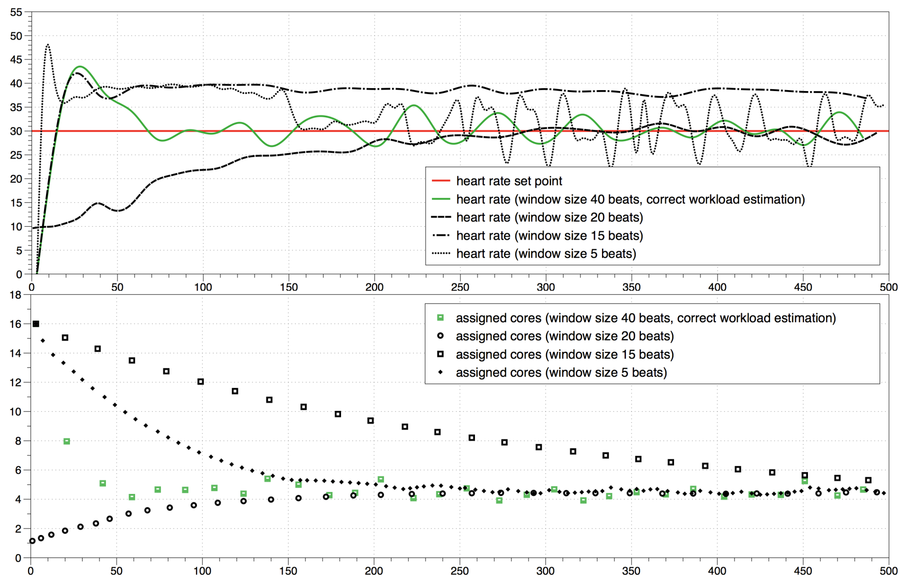

Fig. 4. Results of the first video experiment with x264 when the workload estimation is not correct, in abscissæ the current heart beat. In the first plot, the black curves represent the heart rates with different control periods. In the second plot, the points represent the number of cores assigned before the time division output actuation mechanism. In both the graphs, the green points refer to the curve when the workload is correctly estimated and the period is 1.33s, corresponding to $w_s = 40$.

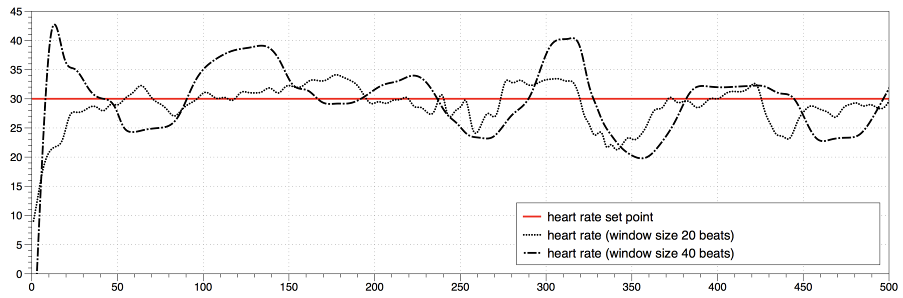

Fig. 5. Results of the second video experiment with X264, in abscissæ the current heart beat, while the black curves depict the heart rate with different control periods.

[3] W. Qin and Q. Wang, "Feedback performance control for computer systems: an LPV approach," in *Proceedings of the 2005 American Control Conference*, vol. 7. Portland, Oregon: IEEE Control Systems Society, June 2005, pp. 4760–4765.

[4] M. Maggio and A. Leva, "A new perspective proposal for preemptive feedback scheduling," *International Journal of Innovative Computing, Information and Control*, (to appear).

[5] H. Hoffmann, J. Eastep, M. Santambrogio, J. Miller, and A. Agarwal, "Application heartbeats: A generic interface for expressing performance goals and progress in self-tuning systems," in *Proceedings of SMART10*, 2010, http://groups.csail.mit.edu/carbon/heartbeats.

[6] C. Karamanolis, M. Karlsson, and X. Zhu, "Designing controllable computer systems," in *Proceedings of the 10th conference on Hot Topics in Operating Systems*. Berkeley, CA, USA: USENIX Association, 2005, pp. 9–15.

[7] K. Hollot, "Worldwide feedback [ask the experts]," *IEEE Control Systems Magazine*, vol. 28, no. 5, pp. 16–43, 2008.

[8] C. Lu, J. Stankovic, and S. Son, "Feedback control real-time scheduling: Framework, modeling and algorithms," *Journal of Real-Time Systems, Special Issue on Control-Theoretical Approaches to Real-Time Computing*, vol. 23, pp. 85–126, 2002.

[9] S. Bang, K. Bang, S. Yoon, and E. Chung, "Run-time adaptive workload estimation for dynamic voltage scaling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 9, pp. 1334–1347, September 2009.

[10] A. Alimonda, S. Carta, A. Acquaviva, A. Pisano, and L. Benini, "A feedback-based approach to dvfs in data-flow applications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 11, pp. 1691–1704, November 2009.

[11] D. Ardagna, C. Cappiello, M. Lovera, B. Pernici, and M. Tanelli, *Active Energy-Aware Management of Business Process Based Applications*, ser. Towards a Service-Based Internet, Lecture Notes in Computer Science. Berlin, Heidelberg: Springer-Verlag, 2008, vol. 5377, pp. 183–195.

[12] A. Varma, B. Ganesh, M. Sen, S. Choudhury, L. Srinivasan, and B. Jacob, "A control-theoretic approach to dynamic voltage scheduling," in *Proceedings of the 2003 international conference on Compilers, architecture and synthesis for embedded systems*. New York, NY, USA: ACM, 2003, pp. 255–266.

[13] A. Azimi and B. Gordon, "Dynamic processor allocation for multiple rhc systems in multi-core computing environments," in *Proceedings of the 2009 conference on American Control Conference*. Piscataway, NJ, USA: IEEE Press, 2009, pp. 4921–4926.

[14] D. Marpe, T. Wiegand, and G. Sullivan, "The h.264 / mpeg4 advanced video coding standard and its applications," *IEEE Communications Magazine*, vol. 44, no. 8, pp. 134–144, 2006.

[15] S. Banerjee, G. Surendra, and S. Nandy, "Exploiting program execution phases to trade power and performance for media workload," in *Proceedings of the 2004 Asia and South Pacific Design Automation Conference*. Piscataway, NJ, USA: IEEE Press, 2004, pp. 387–389.

[16] Z. Sun and S. Ge, *Switched Linear Systems: Control and Design*. London: Springer-Verlag, 2005.

[17] http://www.videolan.org/developers/x264.html.

[18] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *PACT'08*, 2008.

[19] D. Eager, J. Zahojan, and E. Lazowska, "Speedup versus efficiency in parallel systems," *IEEE Transactions on Computers*, vol. 38, no. 3, pp. 408–423, 1989.