A Comparative Evaluation of Latency-Aware Energy Optimization Approaches in Many-Core Systems

Khalil Esper
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany
khalil.esper@fau.de

Stefan Wildermann
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany
stefan.wildermann@fau.de

Jürgen Teich
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany
juergen.teich@fau.de

Abstract

Many applications vary a lot in execution time depending on their workload. A prominent example is image processing applications, where the execution time is dependent on the content or the size of the processed input images. An interesting case is when these applications have quality-of-service requirements such as soft deadlines, that they should meet as good as possible. A further complicated case is when such applications have one or even multiple further objectives to optimize like, e.g., energy consumption.

Approaches that dynamically adapt the processing resources to application needs under multiple optimization goals and constraints can be characterized into the *application-specific* and *feedback-based* techniques. Whereas application-specific approaches typically statically use an offline stage to determine the best configuration for each known workload, feedback-based approaches, using, e.g., control theory, adapt the system without the need of knowing the effect of workload on these goals.

In this paper, we evaluate a state-of-the-art approach of each of the two categories and compare them for image processing applications in terms of energy consumption and number of deadline misses on a given many-core architecture. In addition, we propose a second feedback-based approach that is based on finite state machines (FSMs). The obtained results suggest that whereas the state-of-the-art application-specific approach is able to meet a specified latency deadline whenever possible while consuming the least amount of energy, it requires a perfect characterization of the workload on a given many-core system. If such knowledge is not available, the feedback-based approaches have their strengths in achieving comparable energy savings, but missing deadlines more often.

2012 ACM Subject Classification Hardware → Power and energy; Hardware → Finite state machines; Computing methodologies → Computational control theory; Computer systems organization → Self-organizing autonomic computing

Keywords and phrases energy optimization, control-theory, timing analysis, soft real-time, dynamic voltage and frequency scaling, finite state machines, multi-core, many-core

Digital Object Identifier 10.4230/OASIcs.NG-RES.2021.1

Category Invited Paper

Funding This work is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research-Foundation) – Project Number 146371743 - TRR 89 Invasive Computing.

1 Introduction

Many applications have different requirements that should be met during their execution on modern many-core platforms. Embedded systems often have energy and temperature constraints due to their limited power budget. Interesting is the case when an application comes with more than one non-functional requirement, e.g., latency, power or energy consumption, or safety requirements. In this paper, we consider image processing applications as a class of streaming applications that often require the results to be ready within a defined latency, and at the same time should consume a minimal amount of energy on a given platform. For instance, self-driving cars should timely detect crossing people, and also consume as little energy as possible.

Application-specific approaches for latency-aware energy optimization statically (at design time) determine or approximate a set of operating points where each is optimized for a specific workload scenario [18]. Then, for each workload scenario, a set of actions is determined (e.g., voltage/frequency and/or selection of the number of cores to execute given workload) in order to achieve a set of requirements (e.g., a deadline) while optimizing a cost function, e.g., energy. For example, [6] uses a profiling stage to determine a set of Pareto points. While such approaches rely solely on design-time techniques, other approaches like [9] build an offline model from the profiling data and then adapt it online using machine learning algorithms. However, the disadvantage of these approaches is their lack of ability to adapt to unseen workloads or, more severe, new applications that were not analyzed at the offline stage [9]. The authors of [8] use performance and analytical models online to evaluate a machine learning strategy that was derived offline, in order to adapt to unseen workloads. However, it still needs a profiling stage offline to decide an initial strategy and also cannot be generalized as it depends on the analytical models that are used online.

As a remedy, approaches applying control theory to adapt the execution of applications to current needs have been proposed [3, 7, 10, 11, 12, 13, 16]. Control-theoretical approaches apply an *observe-decide-act* loop, which is a three-stages control strategy. First, the system monitors the application to obtain a latency feedback, then it decides on the actions to be applied by computing a generic control signal that reflects the control system's status and the obtained feedback, and finally it applies those decided actions. The main advantage of these approaches is that the system no longer needs to know the expected workloads or analyze the applications that it will control before start of the execution. Another advantage for using control theory is that if the system can be proven to be *stable*, this implies that the system output will be bounded or return to a desired value in a bounded amount of time [1].

As an alternative to control-theoretic approaches, with equally strong mathematical soundness and proof qualities, finite state machines (FSMs) can be used. FSMs are heavily used in digital design, industrial control and robotics. As such, stability properties might equally be proven by state reachability. Moreover, contrary to many control-theoretic approaches proposed in literature such as [3, 10, 11, 13, 16], no assumptions need to be made on the linearity of as well plant as control behaviours.

Contributions. In this paper, we evaluate and compare both an application-specific [17, 18] and a feedback-based [4] state-of-the-art approach using control theory, and propose a very simple alternative feedback-based approach using FSMs. For streaming-based image processing applications, i.e., video sequences with content-based workload, we subsequently compare these approaches in terms of deadline misses and energy savings for a given many-

core platform. We conclude that application-specific approaches have their strength in case of highly predictable workload whereas the feedback-based approaches have their strength in rather unpredictable and even to some degree to so far unseen workloads.

The remaining of this paper is structured as follows. Section 2 introduces the different approaches for latency-aware energy optimization that shall be analyzed and compared. Section 3 introduces the workload and experiments as well as results on their evaluation and comparison. Finally, in Section 4, we conclude this work.

2 Latency-Aware Energy Optimization Approaches

In this section, we introduce one application-specific and two feedback-based approaches for optimizing energy while preserving soft deadlines for many-core platforms. A given (soft) deadline for execution of periodic workload, e.g., images, shall be enforced while minimizing the amount of consumed energy. The latter can be influenced by variation of as well the number of cores n as well as by dynamic power management, i.e., voltage/frequency scaling. Modern processors allow to execute a program in m different voltage/frequency configurations. In addition, we consider a strategy called race-to-idle as an example of a heuristic approach that is neither application-specific nor feedback-based. Race-to-idle makes the system execute in the fastest possible configuration (e.g., highest voltage/frequency settings $m_{\max}$ and number of cores $n_{\max}$) in order for the application to finish and the system to become idle as fast as possible. However, this method does not provide energy-optimality as has been shown in [5].

2.1 Application-Specific Approaches

Application-specific approaches which are used here synonymously with offline approaches require a knowledge base so they can choose the best decision in the execution phase based on offline optimizations or offline-learned experience. Typically, execution time characteristics are gathered from profiling data. In the following, we introduce one approach as described in [17, 18] more closely.

2.1.1 Profiling Phase

The first phase aims at parameterizing a latency and an energy model for a given application or a task, which is dependent on one or multiple workload indicators (e.g., the number of features in an image) and the configuration (e.g., voltage/frequency and the number of cores) to execute this workload item, respectively task. The approach in [17, 18] considers a class of video (streaming) applications, in particular an object-detection application, in which the workload of some tasks is dependent on the number of input features i. Apart from the number of features, the execution latency L also depends on the voltage/frequency setting m and the number of cores n used for the calculation.

Let $L(1, 1, m_{max})$ denote the latency for processing one feature on one core in the highest voltage and frequency mode m_{max}. $L(1, 1, m_{\max})$ may be determined by simulatively determining the execution latency of the execution per image for a representative set of input images. Subsequently, the latency estimate per feature is determined for each image by dividing its latency by the number of features i in that image. Alternatively, the latency could be determined by applying worst-case timing analysis.

2.1.2 Pareto-Front Determination Phase

After determining $L(1, 1, m_{max})$, a model for estimating the latency $L(i, n, m)$ for processing i features in an image when employing n cores and running in voltage/frequency setting m is derived or learned, resulting in a mathematical characterization of the latency in dependence of workload i and processor setting n and m, e.g.:

$$L(i, n, m) = L(1, 1, m_{\max}) \cdot \frac{i}{n \cdot eff(n)} \cdot \frac{f(m_{\max})}{f(m)} \tag{1}$$

In Eq. (1), $eff(n)$ denotes the parallel efficiency in dependence of the number of cores n employed for the computation with $eff(n) = 1$ in the best case and $f(m)$ being the frequency of setting m. In our experiments described in Section 3, we consider $eff(n) = 1$. Then, based on a given latency deadline $\bar{L}$, a design space exploration (DSE) can then be performed [18] to determine those settings n, m that enable to process the workload i within the deadline $\bar{L}$ while consuming the least amount of energy.

2.1.3 Energy-Minimized Timing Enforcement

Finally, the Pareto-optimal settings can be stored in a table or implemented by an automaton that just selects at run-time the energy minimal setting in characterized ranges of inputs i. Before each execution, based on i, the pre-determined energy-minimal setting $\langle n, m \rangle$ is activated as action. Obviously, this hybrid technique must be adjusted to each application and each architecture individually. However, as we will show in our experiments, if the characterization is *safe*, then we may guarantee deadlines to be enforced by 100%. This means that even hard deadlines can be safely enforced.

2.2 Feedback-Based Approaches

These approaches utilize a latency feedback to decide the next configuration (i.e., $\langle n, m \rangle$) to apply. The most prominent approaches are based on control-theory or finite state machines (FSMs).

2.2.1 Using Control Theory

Using control theory for designing adaptive systems [1] has the advantage of formally guaranteeing the properties (e.g., stability) of systems with unknown workloads [19]. In the following, we describe one concrete approach [4] for subsequent evaluation in more detail. Figure 1 illustrates the overall control mechanism.

Control Signal as an Abstraction for Latency

Control-theory based approaches consider latency as a variable that is controlled, where the controller utilizes Eq. (2) to model the relation between the latency $L(t)$ at iteration t (respectively, discrete time index in the following), and the control signal $s(t-1)$ [4, 11].

$$L(t) = \frac{1}{b(t) \cdot s(t-1)} \tag{2}$$

In Eq. (2), the *base signal* $b(t)$ is a time-varying parameter that forms an abstraction of $L(i, n_{\min}, m_{\min})$, which is the latency of executing the application's workload (i.e., number of features i) at iteration t in the lowest configuration (i.e., $\langle n_{\min}, m_{\min} \rangle$). The base signal is computed using Eq. (3).

$$b(t) = \frac{1}{i(t) \cdot L(1, n_{\min}, m_{\min})} \tag{3}$$

Here, $i(t)$ denotes the number of features i observed at iteration t and $L(1, n_{\min}, m_{\min})$ is the latency of executing one unit of workload (i.e., one feature or $i = 1$) at iteration t in the lowest configuration (i.e., $\langle n_{\min}, m_{\min}\rangle$). As $L(i, n_{\min}, m_{\min})$ generally cannot be known until run time, it is estimated using a Kalman filter [4], or in our use case and later experiments, we use a profiling phase to determine $L(1, n_{\min}, m_{\min})$ for its estimation.

Configuration Mapping

A *configuration mapping* is used in a later energy optimization step to determine the best configurations $\langle n, m\rangle$ for the computed control signal $s(t)$ [4]. This is done by creating a table with an index $s_{n,m}$ that delivers a corresponding configuration $\langle n, m\rangle$ with $s_{n,m}$ being defined as the speedup when executing one unit of workload (i.e., one feature or $i = 1$) in the configuration $\langle n, m\rangle$ over the case when executing this workload unit in the lowest configuration $\langle n_{\min}, m_{\min}\rangle$:

$$s_{n,m} = \frac{L(1, n_{\min}, m_{\min})}{L(1, n, m)} \tag{4}$$

Here, $L(1, n, m)$ denotes the latency of executing one unit of workload (i.e., one feature or $i = 1$) at iteration t in the configuration $\langle n, m\rangle$.

Computing the Control Signal

At iteration t, the controller uses Eq. (5) to calculate the error between the latency in the last execution $L(t-1)$ and the latency goal $\bar{L}$ [4]. In case of a positive error value, the deadline has therefore been missed (i.e., $\bar{L} < L(t-1)$).

$$e(t) = \frac{1}{\bar{L}} - \frac{1}{L(t-1)} \tag{5}$$

The control signal $s(t)$ is then computed using Eq. (6) [4]:

$$s(t) = s(t-1) + (1-p) \cdot \frac{e(t)}{b(t)} \tag{6}$$

Finally, the control signal $s(t)$ is used by the optimizer to find the best configuration (i.e., $\langle n, m\rangle$) that is needed to be applied in the next iteration t, based on the latency error $e(t)$, the base signal $b(t)$, and the previous control signal $s(t-1)$. The base signal $b(t)$ is computed using Eq. (3) after determining the current workload (i.e., number of features $i(t)$). The *pole* p is a user-specified parameter that lies between 0 and 1. A small value of p increases the importance of the error to the resultant control signal, whereas a large p increases the controller resistance to fast-changing workloads.

Energy Optimization

The optimizer uses the configuration mapping, explained above, for transforming the calculated control signal $s(t)$ into the best configuration (i.e., $\langle n, m\rangle$) to meet a given deadline $\bar{L}$, while minimizing the amount of consumed energy. Kim et al. [5] claim that there must be an optimal solution that has at most two configurations that have to be scheduled within the time interval between iteration t and $t+1$. These two configurations are computed using the algorithm which is detailed in [4].

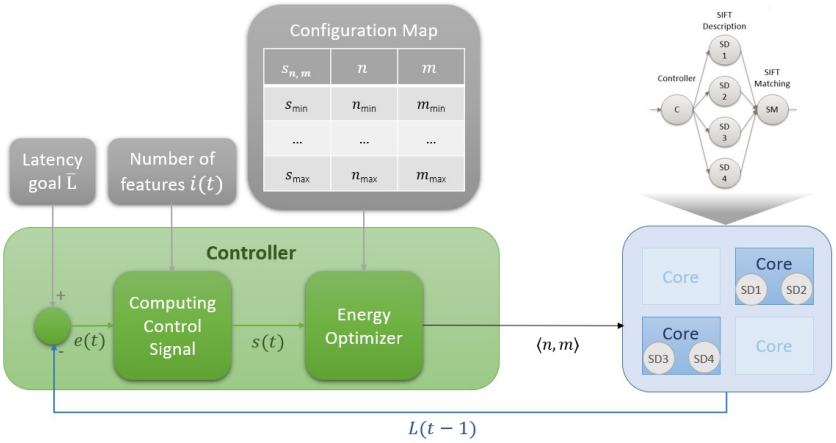


Figure 1 The controller uses the error $e(t)$ and the base signal $b(t)$ (based on the number of features $i(t)$) to compute the control signal $s(t)$. The optimizer looks up the configuration mapping for deciding the next actions (i.e., $\langle n, m \rangle$) based on the computed control signal $s(t)$; the value of $s(t)$ is compared with the values $s_{n,m}$ of the configuration mappings to select the configurations under which to operate the system in the time interval between iteration t and $t+1$[1].

2.2.2 Using Finite state machines (FSMs)

An FSM describes the system's behavior using states, transitions, events, and actions. An FSM is composed of a number of states. It starts in an initial state. Each state outputs actions to perform the desired control activities in case of a Moore machine. Triggered by events (input), it transitions to a next state based on the input and the current state.

FSMs are heavily used for modeling the power states of modern processor architectures including dynamic power management [20]. FSMs have also been proposed for adjusting the processor power depending on a utilization feedback. For instance, the authors of [2] propose an FSM-based controller that uses the processor utilization as a feedback to choose the best configuration (i.e., $\langle n, m \rangle$). Here, the FSM transitions between the states based on utilization thresholds.

A possible realization of an FSM enforcing latency while minizing energy based on feedback is depicted in Figure 2. The error $e(t-1)$ is computed similar to Eq. (5) from the latency feedback $L(t-1)$. This approach neither includes an offline stage nor utilizes a control signal. Instead, it uses a power-ascending list of configurations so that the configuration $\langle n_{c_j}, m_{c_j} \rangle$ associated with state c_j only should have a higher power consumption than that of state c_{j-1}. With a number of N available configurations, the FSM consists of states c_j with $1 \leq j \leq N$. Based on the error $e(t-1)$, there are three possibilities to define the FSM transitions:

1. The latency error is positive $e(t-1) > 0$: The deadline has been missed and the FSM responds by switching to the next state with higher power by incrementing the configuration level.
2. There is no latency error $e(t-1) = 0$: The deadline has been met precisely and the FSM stays in the same state with the same configuration level.
3. The latency error is negative $e(t-1) < 0$: The application executes faster than needed and the FSM responds by switching to the next state with lower power to decrement the configuration level.

[1] In [4], it is explained in detail how a two-step sequence of two $s_{n,m}$ configurations is activated and optimized so to save energy.

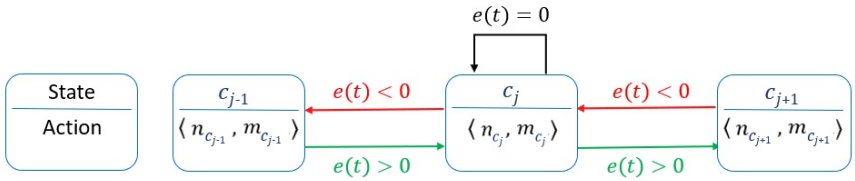


Figure 2 A feedback-based FSM approach, that transitions between adjacent states at each iteration t, based on the feedback $L(t-1)$ and a power-ascending list of configurations.

For the construction of such enforcement FSM, many extensions are possible, e.g., transition conditions containing multiple events (not only one like $e(t)$), possibly also including environmental conditions such as temperature. Moreover, instead of just transiting to one lower, resp. one higher power state, one could transit to non-neighbor states depending on the absolute deviation $e(t)$.

3 Evaluation

3.1 InvadeSIM simulator

For the following evaluations and comparisons, a simulation framework called InvadeSIM, a many-core simulator for parallel applications [15] is used. InvadeSIM allows to specify symmetric homogeneous, as well as tile-based asymmetric many-core architectures within one framework. It performs a discrete-event simulation of applications mapped to a given architecture model including the processor cores, memory access overheads and communication latencies, e.g., on a network-on-chip. In addition, it provides an emulation for the runtime system and a timing as well as a power model for each core type of a heterogeneous many-core architecture. The discrete-event simulation enables the timed simulation of the different parallel tasks in execution. From a power configuration and the execution times, the energy consumption of application executions can be monitored as well. Using an ActorX10 object-oriented programming library [14], applications are modeled by a graph of actors and then mapped and executed on a modeled multi-core platform.

3.2 Object Detection Application

For our evaluations and comparisons, we introduce an object detection application as shown in Figure 3. It belongs to the class of image processing applications that performs a pipelined processing of input image streams. The job of the object detection algorithm is to detect a given object in each image frame by applying a SIFT feature matching algorithm.

The application consists of an actor chain. Each actor processes one input image at a time. The image source (IS) actor reads in the input images periodically at a constant rate, then follows the gray-scale conversion (GS) actor, and after that the edge detection (ED) and the corner detection (CD) actors to determine respectively edges and corners in an image. After that the SIFT orientation (SO) actor achieves invariance to image rotation. The four SIFT description actors SD_1 to SD_4 extract the features in an image. They can be executed in parallel on $n = 4$ cores, after partitioning the number of features i of a given image evenly into each actor.

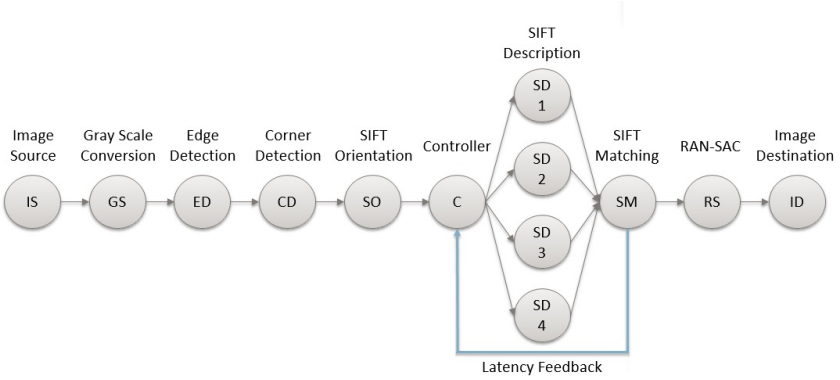


Figure 3 Object detection algorithm implemented as a graph of actors for pipelined processing of streams of images.

3.3 Target Architecture and Deadline Model

For the following experiments, let each of the periodic execution of each SD actor be completed within a soft deadline of $\bar{L} = 80$ ms. For the enforcement of this local deadline, the execution power mode m (voltage/frequency) of the SD actors' cores through Dynamic Voltage and Frequency Scaling (DVFS) is used and we assume, a maximum of $n = 4$ cores can be activated in each of $m = 20$ different power modes. However, during the execution of an image, we assume all cores run in the same power mode m, thus resulting in a configuration $\langle n, m \rangle$. According to Figure 3, the SD actor also provides a feedback of the latency which can be used by the controller to properly determine the configuration $\langle n, m \rangle$ to be used to execute the next frame, resp. iteration. Upon each execution, the output of each SD actor is then provided to the SIFT matching (SM) actor to detect common features between the given object to be found and the current input image. Then, the RAN-SAC (RS) actor calculates the transformation between both images based on the matched features. The image is finally sent out by an image destination (ID) actor.

3.4 Application Management Techniques

In the following, we evaluate four techniques for latency-aware energy optimization:

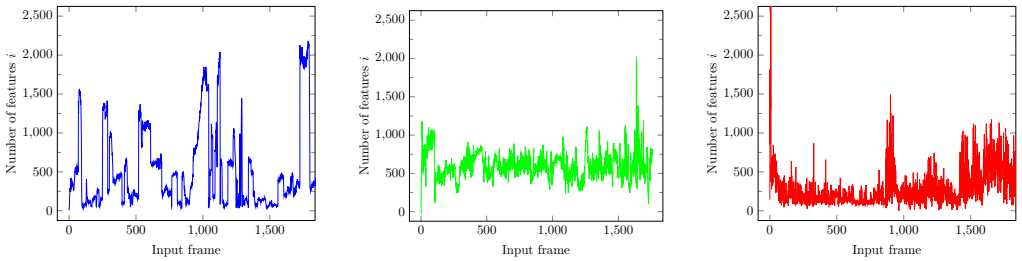
1. Race-to-idle: Executing always in the highest configuration level $\langle n_{\max}, m_{\max} \rangle$.
2. Application-specific: Finding operating points offline by analyzing sequences of input images beforehand, see Section 2.1. At run-time, the Pareto-optimal configuration to enforce the given deadline is retrieved from a table based on the characterized input (feature number i) [17, 18].
3. Control-theoretical: Computing a control signal that translates into a specific configurations $\langle n, m \rangle$ based on latency feedback $L(t-1)$ [4].
4. FSM-based: Using the simple FSM from Figure 2 that is based on latency feedback $L(t-1)$ to transition between neighbor system configurations $\langle n, m \rangle$.

3.5 Workload Types

The following experiments have been performed by applying the following workloads to the object detection application with latency-enforced SD actor:

- A 5-minute YouTube video from national geographic.
- A 40-minutes video of front camera in a car driving through city roads.
- A sequence of $1,750$ images with random contents.

Figure 4 shows the number of features i for each image of the sequences.



(a) YouTube video. **(b)** Car video. **(c)** Random images.

Figure 4 Distribution of number of features i per frame in the analyzed videos.

3.6 Results

We run the application for each combination of application management technique and input sequence. Each controller behavior has been implemented by the controller actor in Figure 3. The number of available configurations is $N = n \cdot m$ with $n = 4$ and $m = 20$. As latency bound $\bar{L}$ of the enforced SD actor, we chose $\bar{L} = 80\ ms$. For the control-theoretic approach, the pole was chosen at $p = 0.5$ (changing the pole value did not have a noticeable impact on its results).

In the following, we analyze the performance of the four approaches in terms of number of deadline misses and in terms of energy consumption.

3.6.1 Deadline Misses

Mean Absolute Percentage Error (MAPE) is a standard metric in controllers [4], and we use it to measure number of deadline misses. We can compute MAPE for an application with K iterations (number of frames in our test applications) using Eq. (7):

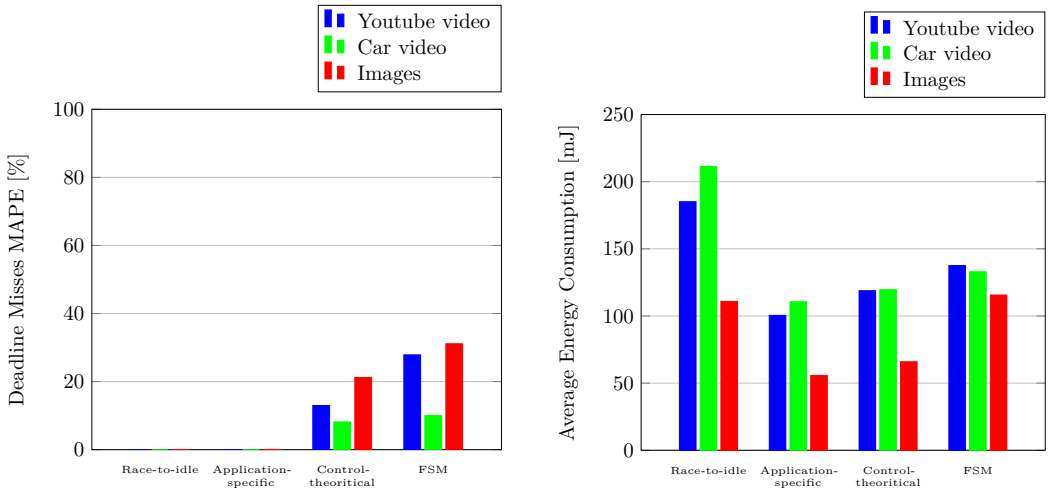
$$MAPE = 100\% \cdot \frac{1}{K} \sum_{i=1}^{K} \begin{cases} L(t-1) > \bar{L} : \dfrac{L(t-1) - \bar{L}}{\bar{L}} \\ L(t-1) \leq \bar{L} : 0 \end{cases} \tag{7}$$

Figure 5a shows the number of deadline misses experienced for each of the four evaluated approaches. Race-to-idle and the application-specific approach almost never miss any deadline as they execute in the fastest possible way, respectively the slowest required speed to meet the deadline. Only in a few cases where the deadline cannot be met at all even in the fastest possible configuration, $\bar{L}$ is (necessarily) exceeded. However, evidently, the feedback-based approaches miss more deadlines, especially the FSM-based approach, which obviously by its simple construction provides only a stepwise and thus slow convergence towards a feasible level of configuration $\langle n, m \rangle$, after receiving the latency feedback $L(t-1)$.

3.6.2 Energy Consumption

Figure 5b shows the evaluated average overall energy consumption per image of the SD actor. Race-to-idle consumes the largest amount of energy, because it executes in the fastest possible way. The application-specific approach consumes a significantly less amount of energy, due to the fact that it knows exactly which actions are needed to meet the deadline for all possible

input images. For the feedback-based approaches, the control-theoretical approach consumes slightly more energy than the optimal (i.e., the application-specific approach). On the other hand, the simple FSM-based approach consumes more energy than the application-specific and the control-theoretical approach due to its simplicity in construction. Still, it consumes less energy than the race-to-idle strategy.



(a) Percentage of deadline misses of each approach.

(b) Energy consumption of each approach.

Figure 5 Evaluation results of the race-to-idle, application-specific and feedback-based approaches using the three types of image input sequences and a set of $N = 80$ configurations in terms of deadline misses (a) and consumed energy (b).

3.6.3 The Effect of Available Configurations

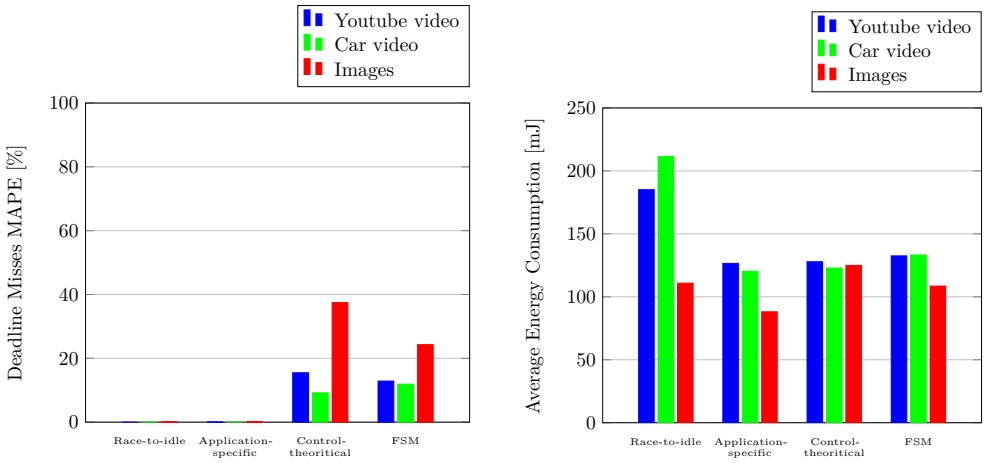
In a further evaluation, we fixed the number of cores to $n = 4$, resulting into only $N = 20$ configurations (i.e., 20 voltage/frequency settings), keeping race-to-idle unaffected.

We notice from the results in Figure 6 an increase in the amount of energy consumption for the application-specific and the control-theoretical approach. For the latency results, the FSM-based approach achieves less deadline misses compared to the case of $N = 80$, because now, the transitions in the FSM are more effective in the sense of faster converging to a feasible configuration level $\langle n, m \rangle$.

4 Conclusion and Future Work

In this paper, we evaluated and compared multiple application-specific and feedback-based categories for latency-aware energy optimization. We evaluated the approaches on a many-core simulator and found out that the application-specific approach respects the latency goal whenever possible while consuming the lowest amount of energy consumption. For the feedback-based category, the feedback-based approaches can achieve energy savings comparable to the application-specific approach, but both missing deadlines more often.

For future work, we aim to focus on FSM-based approaches further, as they are similar to control-theoretical approaches in providing a sound mathematical formalism with opportunities of formally proving requirements on non-functional program properties, e.g., by state reachability and the use of temporal logic to express complex requirements. On



(a) Percentage of deadline misses of each approach.

(b) Energy consumption of each approach.

Figure 6 Evaluation results of the race-to-idle, application-specific and feedback-based approaches using three types of image input sequences and a set of $N = 20$ configurations in terms of deadline misses (a) and consumed energy (b).

the other hand, control-theoretic approaches, although theoretically sound and known for their strength in being able to mathematically prove properties such as the stability and robustness of a feedback-based system, are often based on assumptions of linearity of either controller or the many-core system under control, e.g., based on z-transform descriptions and the analysis of poles of related closed-loop transfer functions. However, in multi-core systems, performance such as the speedup hardly scales linearly with the number of cores for most workloads and applications. Therefore, non-linear control techniques would need to be applied.

References

1 Joseph L Hellerstein, Yixin Diao, Sujay Parekh, and Dawn M Tilbury. *Feedback control of computing systems*. John Wiley & Sons, 2004.

2 Sebastian Herbert and Diana Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *Proceedings of the 2007 international symposium on Low power electronics and design (ISLPED'07)*, pages 38–43. IEEE, 2007.

3 Henry Hoffmann, Martina Maggio, Marco D Santambrogio, Alberto Leva, and Anant Agarwal. A generalized software framework for accurate and efficient management of performance goals. In *2013 Proceedings of the International Conference on Embedded Software (EMSOFT)*, pages 1–10. IEEE, 2013.

4 Connor Imes, David HK Kim, Martina Maggio, and Henry Hoffmann. Poet: a portable approach to minimizing energy under soft real-time constraints. In *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 75–86. IEEE, 2015.

5 David HK Kim, Connor Imes, and Henry Hoffmann. Racing and pacing to idle: Theoretical and empirical analysis of energy optimization heuristics. In *2015 IEEE 3rd international conference on cyber-physical systems, networks, and applications*, pages 78–85. IEEE, 2015.

6 Zhiquan Lai, King Tin Lam, Cho-Li Wang, and Jinshu Su. Latency-aware DVFS for efficient power state transitions on many-core architectures. *The Journal of Supercomputing*, 71(7):2720–2747, 2015.

7 Martina Maggio, Alessandro Vittorio Papadopoulos, Antonio Filieri, and Henry Hoffmann. Automated control of multiple software goals using multiple actuators. In *Proceedings of the 2017 11th joint meeting on foundations of software engineering*, pages 373–384, 2017.

8 S. K. Mandal, U. Y. Ogras, J. Rao Doppa, R. Z. Ayoub, M. Kishinevsky, and P. P. Pande. Online adaptive learning for runtime resource management of heterogeneous socs. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020. doi:10.1109/DAC18072.2020.9218604.

9 Sumit K. Mandal, Ganapati Bhat, Janardhan Rao Doppa, Partha Pratim Pande, and Umit Y. Ogras. An energy-aware online learning framework for resource management in heterogeneous platforms. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 25(3):1–26, 2020.

10 Asit K Mishra, Shekhar Srikantaiah, Mahmut Kandemir, and Chita R Das. Cpm in cmps: Coordinated power management in chip-multiprocessors. In *SC'10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE, 2010.

11 Nikita Mishra, Connor Imes, John D Lafferty, and Henry Hoffmann. Caloree: Learning control for predictable latency and low energy. *ACM SIGPLAN Notices*, 53(2):184–198, 2018.

12 Amir-Mohammad Rahmani, Mohammad-Hashem Haghbayan, Anil Kanduri, Awet Yemane Weldezion, Pasi Liljeberg, Juha Plosila, Axel Jantsch, and Hannu Tenhunen. Dynamic power management for many-core platforms in the dark silicon era: A multi-objective control approach. In *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 219–224. IEEE, 2015.

13 Karthik Rao, Jun Wang, Sudhakar Yalamanchili, Yorai Wardi, and Ye Handong. Application-specific performance-aware energy optimization on android mobile devices. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 169–180. IEEE, 2017.

14 Sascha Roloff, Frank Hannig, and Jürgen Teich. ActorX10 and run-time application embedding. In *Modeling and Simulation of Invasive Applications and Architectures*, pages 129–164. Springer, 2019.

15 Sascha Roloff, Frank Hannig, and Jürgen Teich. *Modeling and Simulation of Invasive Applications and Architectures*. Springer, 2019.

16 Stepan Shevtsov and Danny Weyns. Keep it simplex: Satisfying multiple goals with guarantees in control-based self-adaptive systems. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 229–241, 2016.

17 Jürgen Teich, Behnaz Pourmohseni, Oliver Keszocze, Jan Spieck, and Stefan Wildermann. Run-time enforcement of non-functional application requirements in heterogeneous many-core systems. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 629–636. IEEE, 2020.

18 Jürgen Teich, Pouya Mahmoody, Behnaz Pourmohseni, Sascha Roloff, Wolfgang Schröder-Preikschat, and Stefan Wildermann. Run-Time Enforcement of Non-functional Program Properties on MPSoCs. In Jian-Jia Chen, editor, *A Journey of Embedded and Cyber-Physical Systems*. Springer, 2020. doi:10.1007/978-3-030-47487-4.

19 Danny Weyns, M Usman Iftikhar, Didac Gil De La Iglesia, and Tanvir Ahmad. A survey of formal methods in self-adaptive systems. In *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*, pages 67–79, 2012.

20 Yang Xu, Rafael Rosales, Bo Wang, Martin Streubühr, Ralph Hasholzner, Christian Haubelt, and Jürgen Teich. A very fast and quasi-accurate power-state-based system-level power modeling methodology. In Andreas Herkersdorf, Kay Römer, and Uwe Brinkschulte, editors, *Architecture of Computing Systems - ARCS 2012 - 25th International Conference, Munich, Germany, February 28 - March 2, 2012. Proceedings*, volume 7179 of *Lecture Notes in Computer Science*, pages 37–49. Springer, 2012. doi:10.1007/978-3-642-28293-5_4.