All-digital control-theoretic scheme to optimize energy budget and allocation in multi-cores

Davide Zoni*, Luca Cremona, and William Fornaciari, *Senior Member, IEEE*

Abstract—The Internet-of-Things (IoT) revolution fueled new challenges and opportunities to achieve computational efficiency goals. Embedded devices are required to execute multiple applications for which a suitable distribution of the computing power must be adapted at run-time. Such complex hardware platforms have to sustain the continuous acquisition and processing of data under severe energy budget constraints, since most of them are battery powered. The state-of-the-art offers several ad-hoc contributions to selectively optimize the performance considering aspects like energy, power, thermal or reliability. However, there is a need for a generic coordinated management strategy able to cope with all of these dimensions, while allowing the Operating System (OS) and the applications to "suggest" or constrain the actuation. This paper proposes a unified control-theoretic scheme to coordinate the design of energy-budget and energy allocation solutions for multi-cores. The proposed controller can work with any actuator and it can interact, at run-time, with both the applications and the OS to optimize the actuation signals steering the computing platform. Such control scheme offers the possibility to integrate any performance related policy in the form of an energy-allocation strategy, still ensuring the theoretic exponential stability of the overall controller if the actuation of the policy, coming from the OS and the applications, "is not too fast". To demonstrate the feasibility of our solution, we implemented the controller into a RISC multi-core running on the Xilinx Artix 100t FPGA device, available in the the Digilent Nexys4-DDR board. Results considering two actuators and both the quad- and the eight-core version of the considered computing platform, highlight the scalability of the proposed solution as well as an area overhead for the –all digital, on chip– controller limited to 0.86% (FFs) and 5.3% (LUTs) of the FPGA chip. We also considered a dynamic scenario validating the speed of the controller, where our framework has to face with modifications to the energy-allocation control policy carried out by the OS and the applications. The obtained results are collected by executing a huge mix of benchmarks and the statistical significance is accounted by executing each scenario 30 times. Such results are analyzed considering three quality metrics. First, the efficiency in exploiting the imposed budget (EFF_g) that is on average 98.27%. Second, the overflow of the actual average power consumption with respect to the assigned budget (OVF_g), which is limited to 1.43 mW on average. Last, the performance utility loss due to the control scheme that is limited to 1.87% on average.

Index Terms—energy-budget, microarchitecture, multi-core, IoT, power-performance optimization, control-theoretic scheme, efficient computing.

1 Introduction

Energy efficiency represents a standing obstacle for the evolution of any computing platform, limiting the performance of both embedded and high-performance computing (HPC) applications. Unfortunately, the practice of minimizing the energy consumption under performance constraints, commonly used in HPC scenarios, cannot be readily applied to embedded platforms. In fact, the latter, which are battery-powered, exhibit an opposite optimization problem that imposes to maximize the performance under energy-budget constraints since they are, most of the time, battery-powered. Moreover, the traditional embedded platforms, that were in charge of a single task, have been replaced by complex multi-cores executing multiple concurrent applications where specialized hardware accelerators are used to offload the most intensive part of the computation. The complexity of such architectures, coupled with the required computational efficiency, highlights a coordinated management problem made of two tightly linked aspects. First, an energy-budget management strategy is required to ensure that the overall device remains operational for a given amount of time. Second, an allocation scheme is required to split the energy budget inside of the computing platform, also accounting for the application-level requirements. However, a single policy cannot easily encompass both the energy-budget and the application-specific requirements.

In fact, the state-of-the-art proposals rely on ad hoc-solutions to solve specific energy-performance optimization problems. There exists some control-theoretic PID schemes used to trade performance with energy [1], thermal [2] or reliability [3] aspects. Such schemes ensure theoretical stability for the controller that, however, is too simple to manage either application constraints or multiple system-wide objectives. In contrast, the majority of the investigations in the state-of-the-art deliver heuristic, or algorithmic implementations, for which even no theoretical guarantee on stability or optimality can be ensured. In general, the use of ad-hoc schemes prevents any comparative analysis and any reuse in application scenarios that are even slightly different from the original one. The majority of the proposals leverages Dynamic Voltage and Frequency Scaling (DVFS) as the sole actuator to achieve both the energy-budget and the performance goals. We note that the effectiveness of the

* *Corresponding and leading author: Davide Zoni*

• *D. Zoni, L. Cremona and W. Fornaciari were with the Dipartimento di Elettronica Informazione e Bioingegneria, Politecnico di Milano, 20133 Milano, Italy.*
E-mail: {davide.zoni,luca.cremona,william.fornaciari}@polimi.it

Manuscript received April 22, 2019; revised XXX.

DVFS mechanism is mitigated by its complex mixed analog-digital design that imposes the use of voltage and frequency islands and the use of a proper resynchronization logic infrastructure [4]. To this extent, faster and simpler actuators are emerging to support the run-time optimizations in the embedded platforms domain. To limit the design complexity of the power rails to support hardware reconfigurability, Xilinx FPGAs only offers Dynamic Frequency Scaling (DFS) only. Moreover, [1] proposed a Dynamic Clock Gating (DCG) actuator that avoids signals resynchronization and further increases the dynamic of the actuator.

On a different but still related perspective, the use of software-implemented energy-budget and energy-allocation strategies, represents a viable solution for HPC and server systems. Nevertheless, it can be detrimental for the performance of embedded platforms, since part of the computing resources must be devoted to the computation of both the power estimate and the control action. Recent investigations [5], [6], [7] proposed all-digital power monitoring solutions that estimate the consumed power from the switching activity of selected signals at the microarchitectural level. However, there is a complete absence of a unified and general methodology to control different energy related aspects, also considering applications and Operating System requirements.

Contributions - This work presents a control-theoretic scheme to design coordinated energy-budget and energy-allocation solutions for multi-cores. The controller has been implemented in hardware and assessed on 4- and an 8-core processors via the Digilent Nexys4-DDR board featuring a Xilinx Artix7 100t FPGA device. To demonstrate the effectiveness of the proposed solution, we employed the online power monitoring infrastructure presented in [6] and the power-cap PID controller presented in [1]. In particular, the proposed multi-level controller advances the state-of-the-art in three main directions:

- Generic coordinated control-theoretic scheme - The proposed solution is neither an ad-hoc algorithm nor an heuristic, but a complete framework to design generic energy-budget and energy-allocation controllers for multi-cores. The controller is meant to work with any actuator and the exponential stability property of the final controller has been demonstrated theoretically and validated experimentally.
- Manage OS and application constraints - The controller can seamlessly integrate energy-budget and energy-allocation constraints coming from the applications and the Operating System, thus addressing the ever increasing popular run-time frameworks that support the self-adaptive application paradigm [8].
- All-digital and lightweight scheme - An instance of the proposed controller has been implemented in hardware using 4- and 8-core processors to demonstrate its feasibility and scalability. The standard logic digital-only hardware implementation on a real FPGA prototype, highlights the wide applicability of the controller, with no impact on the timing and with a very low area overhead.

The rest of this manuscript is organized in four sections. Section 2 details the background and the state-of-the-art on energy-budget and energy allocation. The proposed control-theoretic scheme is discussed in Section 3. Section 4 presents the microarchitecture and the results considering a quad- and an eight-core processor. Conclusions and future works are drawn in Section 5.

2 Background and Related Works

Figure 1 shows the block diagram of a general energy-budgeting and energy-allocation solution, as made of four parts: *i)* the online power monitor, *ii)* the actuator, *iii)* the interface between the controller and the applications or the OS, and *iv)* the controller. The power monitor periodically generates the estimate of the power consumption starting from the collected statistics of the architectural behavior (see *Monitored statistics* in Figure 1). The power estimates as well as the system and application constraints are then fed into the controller that generates the control signal for the actuator, which is responsible for driving the energy knobs of the platform. Note that the state-of-the-art proposes high-level run-time management frameworks where either a run-time manager [9] or the applications themselves [8], can read-out the power consumption to later suggest or constrain the controller actuation (see app/system constraints in Figure 1).

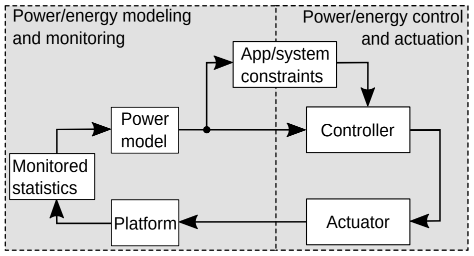


Fig. 1: Scheme of a generic energy-budgeting and energy allocation control-loop encompassing the online power monitor, the controller, the power actuator and the constraints coming from the OS and the applications.

The rest of this section details the state-of-the-art solutions for each part of the general control loop shown in Figure 1. The focus is on the possibility and limitations of designing a generic control-based framework to work with different actuators and that can seamlessly consider constraints and requests from both the OS and the applications. The final goal of this work is the design of a generic and flexible controller for multi-cores addressing energy-budget and energy-allocation aspects. The related discussion on actuators and power monitoring schemes is included to provide a comprehensive picture, but it is not a contribution of this work. The power model generates power estimates, each one corresponds to the average power consumption computed in the specific time period. To the sake of readability, the rest of this manuscript refers to power rather than energy (for both estimate and actuation), since the latter can be easily derived from the former, once the time period is fixed.

Online power monitoring schemes - the set of hardware- and/or software-implemented strategies to either measure or to estimate the power consumption of the considered computing platform with a specified timing resolution. In particular, we focus on power estimation, since the direct power measurement requires analog meters and custom logic thus hindering their fine grain integration embedded systems. Due to its flexibility and the ease of integration in complex designs, power estimation represents the de-facto, cost-effective solution to monitor the power consumption in computing devices. Such indirect measurement methods are made of two stages: a design-time part and a run-time part. At design-time, the power model is commonly identified by leveraging the relationship between the power consumption and the internal switching activity of the target architecture. The possibility of using such relationship at different abstraction levels allows a trade-off between the accuracy of the power estimate with the complexity of the power monitoring architecture. At run-time, the implementation of such model is continuously fed with the same architectural statistics to periodically deliver the power estimates. Software-implemented online power monitoring schemes are usually employed for HPC and high-end embedded systems with limited energy constraints. In particular, such approaches leverage the architectural performance counters as the source of the statistics [10]. [11] proposed a power monitoring solution that exploits the CPU utilization statistics as proxies of the switching activity of the microarchitecture. [12], [13], [14] discussed online power monitoring infrastructures that make use of the architectural performance counters or events to identify a power model. Hardware-implemented online power modeling solutions (all-digital) emerged as a viable approach to address strict computing efficiency constraints. In particular, their implementation minimizes the performance overhead typically associated with the software-implemented schemes that, unfortunately, compete with the other applications for the CPU. Hardware solutions can also increase the timing resolution of the entire monitoring system to efficiently manage faster physical phenomena [5], [6], [7]. Note that our proposal targets the design of a controller that leverages on the all-digital scheme proposed in [6] as the online power monitor whose detailed description is out of the scope of this research.

Power actuators - the Dynamic Voltage and Frequency Scaling (DVFS) is commonly employed in HPC and high-end embedded platforms [15], [16] for energy-budgeting and power management. However, its complex mixed analog/digital design can limit its applicability in cost-effective designs, where less complex and cheaper actuators can better satisfy the cost-efficiency trade-off. Dynamic Frequency Scaling (DFS) is cheaper than DVFS, since it does not scale the voltage at run-time. Moreover, the reduced scalability of voltage due to the technology evolution, makes the efficiency of DFS closer to the one of DVFS [17], [18]. In particular, Xilinx FPGAs that are more and more employed in both HPC and embedded projects, only offer DFS actuators. This is due to the inherent complexity of designing power rails for logic blocks that can reconfigure their function, and consequently their power needs, at run-time. Moreover, high-end microcontrollers are not offering DVFS or DFS. Such choice is due to the complexity of designing the resynchronization logic blocks to interface different frequency islands and the need for an analog-digital design, to implement the VCO (Voltage Controlled Oscillator) within both DFS and DVFS actuators. They usually expose only sleep states to save power during the idle periods even if cheap and fast actuators could be of great help to trade power for performance when the platform is active. To this extent [1] proposed the Dynamic Clock Gating (DCG) actuator as a simple yet effective mechanism to trade power for performance at run-time, without resorting to complex resynchronization and mixed analog-digital designs. The DCG works by glitching the clock signal with a net effect of reducing the dynamic power consumption of the target computing platform. In general, to broaden the usability of the controller, any control mechanism addressing energy-budget and energy-allocation problems should not be tailored to a specific actuator.

Application and Operating System (OS) constraints - to maximize the efficiency of the computing platform, OS and applications should be able to "influence" or constraint the decision of the energy-budget/allocation controller. For example, [8] proposed a software-based framework where the applications can monitor their performance and power consumption to later autotune their allocated resources, using a distributed consensus algorithm. [9] presented a run-time resource manager that collects the application requests in terms of Quality-of-Service (QoS) and the platform constraints in terms of available resources and energy-budget, to produce a global allocation of the resources to the running applications. We note that offering a control scheme that can consider both application- and OS-level requirements to generate the actuation signals, is of paramount importance. To this extent, the proposed control scheme can fulfill such requirements by construction, since the OS can specify the energy-budget at run-time and both the OS and the applications can specify the energy-quota for some or all the applications running on specific computing resources of the platform. In particular, we make it possible to constrain the energy quota of some cores while optimizing the remaining ones with a global policy. The particular structure of the proposed control scheme allows to add and to remove such energy-quota constraints at run-time, while still ensuring the exponential stability property of the global controller.

Energy-budget and energy-allocation schemes - The state-of-the-art reports different proposals addressing energy-budget and energy-allocation requirements in HPC scenarios. To optimize the total energy consumption by consolidating multi-tier workloads in data-centers, [19] employs a queuing-theoretic formulation. [20] proposes another queuing-theoretic scheme to design two policies, each one able to address either the energy-budget or the power-cap aspects in server architectures. The work employs the DVFS actuator which has been also largely employed in other control-theoretic approaches [21], [22]. [17] presents a policy to minimize the energy consumption by scaling the CPU speed without violating task deadlines. [23] proposes a policy to balance the energy budget among critical and non-critical tasks. Different state-of-the-art proposals addressed the workload characterization problem to support run-time power-performance optimizations in multicores [24], [25]. [24] proposed to model the application behavior using fractal equations rather than linear ones to capture some important characteristics of the workloads. However, as it is prohibitively complex to discover, for some applications, an accurate model and [24] suggested to resort to statistical methods. We note that the application modeling falls outside the scope of our proposal. However, the proposed controller can seamlessly integrate any application constraints coming from the analysis using an accurate software-level workload model of the running application (see Section 4.4). [26] demonstrates a software approach to optimize at runtime the voltage and frequency level of the Intel SCC. The controller observes the traffic crossing boundaries of different voltage and frequency island to optimize the power-performance trade-off of the computing platform.

Intel RAPL [27] represents the energy-budget and energy-allocation solutions for multi-core proposed by Intel for its own Sandy-bridge processors. RAPL implements a heuristic to manage the budget and the allocation of energy among the CPUs and memory employing DVFS. However, RAPL represents a closed solution specifically tailored to Intel processors for which the design details are not available. Moreover, it does not offer any stability guarantee and it is not meant to seamlessly integrate with OS and application constraints, e.g., self-adaptive application frameworks [8].

In contrast, our proposal presents a framework that can coordinate in novel ways the energy budget and the energy-allocation problems, with three contributions to the state-of-the-art. First, it can seamlessly integrate application and OS requirements and constraints, without losing the stability property. We note that self-adaptive applications, e.g., Margo [8], and centralized run-time resource management schemes, e.g., BBQ [9], can implement any sort of ad-hoc policy for which the computed solution can periodically override either the energy-budget or the energy-allocation or both actions of our controller for part or for all the running applications. Second, our controller can work with any actuator and it is not only meant for DVFS. Third, the standard-cell hardware implementation on a real prototype is feasible and it has low area overhead. This is in net contrast with the majority of the state-of-the-art proposals, that resort either to software-based implementations, or to partial hardware implementations of simple PID solutions. To work, the former requires a full OS stack while for the latter a complete prototype implementation is typically not reported, making difficult to evaluate their scalability and feasibility. Moreover, some solutions are ad-hoc and closed-source, e.g., Intel RAPL, thus not easily employable on different computing platforms.

3 Methodology

This section discusses the proposed all-digital coordinated control scheme to optimize the energy-budget and energy-allocation aspects. The scheme is implemented in the form of a discrete time-domain controller to allow an all-digital implementation. The resolution of the discrete time-domain step is selected equal to 2μs, thus each time epoch k measures a 2μs period of time. Such value is not a limitation of the proposed control scheme; rather it allows to demonstrate the feasibility of an all-digital controller implementation. The proposed controller can work with any coarser time resolution without changing any parameters, since the power estimates would be observed by the controller as a moving average series of values. However, in case a finer time resolution is required for the power estimates, the stability analysis and the performance of the controller must be assessed again. We note that the considered time resolution allows to manage the energy-budget and the energy-allocation problems considered in this work. In summary, to the best of our knowledge, this is the first controller that can work at such time resolution and, at the same time, is able to consider all the goals mentioned above in an integrated manner, providing also a complete hardware implementation.

Figure 2 depicts the hierarchical structure of the proposed control scheme, which has three parts. At the innermost level, a local control loop is implemented for each computing unit (CPU_i) (see *Local control-loop i-th* in Figure 2). The number of implemented CPUs is $nCpu$. The *i-th* local controller of such innermost loop regulates the control action to ensure the power consumption of the corresponding *i-th* core ($P_{k,i}$) to follow the local set point value (see $P_{k,i}^{SP}$ in Figure 2). The outer layer (see *Global control-loop* in Figure 2) is made of a single-input single-output (SISO) *Global controller* that generates a correction factor to the power budget (S_k^{Tot}) starting from the difference between the global set-point $P_k^{Tot,SP}$ and the total consumed power in the considered time epoch k (P_k^{Tot}). We note that the global power set-point ($P_k^{Tot,SP}$) is imposed by either the operating system or the resource manager and it represents the average power consumption to enforce the required energy-budget. Moreover, it can change overtime to honor any higher level policy and requirement.

The correction factor S_k^{Tot} is generated by the global controller and it is added to the P_k^{Tot} quantity to generate the global available power budget ($P_k^{TotCorr,SP}$) for the time epoch k. Such budget is split between the local controllers in the form of their local set-points ($P_{k,i}^{SP}$) (see *Inner control-loop* in Figure 2). We note that the saturation blocks in the *Global control-loop* are used for implementability reasons and their saturating values have been selected according to the employed reference platform (see Section 4.1 for further details). On top of the control loops, the *Supervisor algorithm* actually implements the energy-allocation policy by modifying the θ coefficients of the SIMO box. Such controller regulates the quota of the energy-budget assigned to each local controller in the form of its set-point value. As one of the possible strategies, this manuscript presents a fair energy-allocation strategy for performance balancing. However, we note that the *supervisor algorithm* can implement any scheme or heuristic to shape the performance metric, thus ensuring a great flexibility in the customization of the system.

The rest of this section is organized in three parts. Section 3.1 details the design of the local controller. The design of the global controller is discussed in Section 3.2, while Section 3.3 presents the supervisor algorithm that we adopted in this paper and that represents a possible policy. We note that the controller design is an iterative process, that starts with employing the simplest controller structure, to then increase its complexity and meet the expected goals. As in any control-theoretic design, we started describing the controller design from the inner-most to the outer-most control-loop imposing that the inner-loop has a dynamic

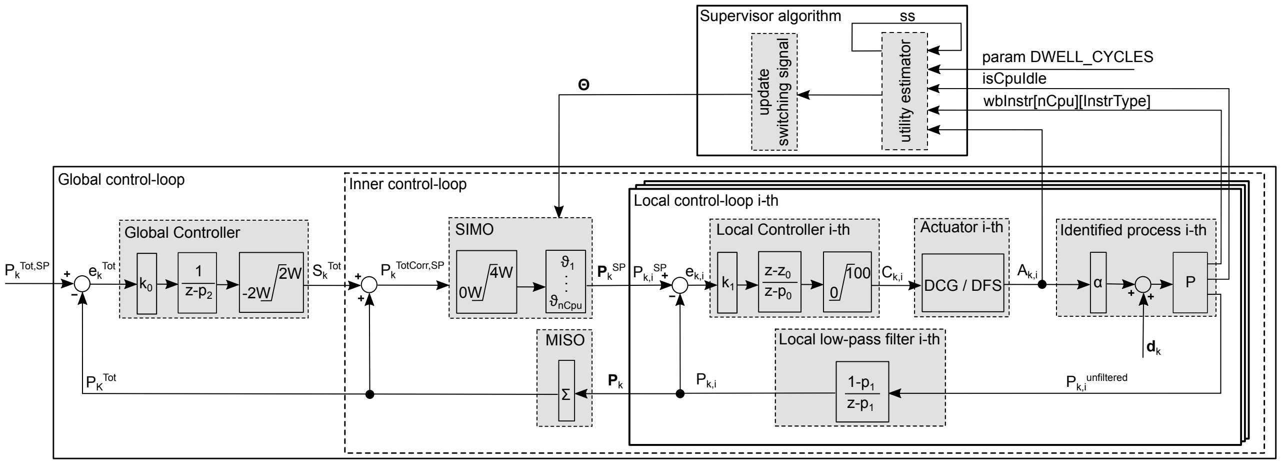


Fig. 2: Closed-loop view of the proposed all-digital coordinated energy-budget and energy-allocation system. Local control-loop parameters: z_0=0.32, p_0=1, p_1=0.5. Global control-loop parameters: p_2=1, k_0=0.01. The identified α is equal to 0.8.

that is at least 10 times faster that the one of the outer control-loop. Moreover, the inner-most controller, i.e., local controller, must follow the given set-point values as close as possible, while the global controller is meant to provide energy-budgeting support. The latter is also in charge of buffering the unused energy quota in each time epoch to make it available in the future if requested by the applications. However, the structure of the controller which offers the buffering capability does not prevent the overflow of the imposed energy-budget at steady-state.

3.1 Local controller design

The design of a local controller takes steps from the work in [1] that proposes a control scheme made of a programmable control-based PID coupled with a low pass filter. Starting from the proposal in [1], this section describes its extension and the resulting final control-loop function useful to support the stability analysis of the cascade control systems proposed in this manuscript.

We implement a local controller for each i-th core of the considered multi-core platform. Each local controller takes in input the energy-budget dynamically imposed by the global controller in the form of a power set point value (P_k, i^{SP}) and actuates on the process P that models the power consumption of the i-th core. We leverage the online power monitoring scheme presented in [6] that generates a power consumption sample every 2μs. The actual power consumption ($P_{k,i}$) is backward propagated via a low pass filter to generate the error signal ($e_{k,i}$) which is then fed to the controller. The control signal ($C_{k,i}$) pushes the actuator to directly impact on the controlled variable, i.e., the power consumption of P, through the actuation $A_{k,i}$.

The d_k quantity models the non-controllable disturbance on the power consumption. We note that the integral formulation that is generally employed to model the power consumption [19], [20] is accurate only if the time-resolution of the system stays in the order of milliseconds. To work at the granularity of microseconds, we employ the power model defined by Eq. 1 ([1]) that models the power consumption as a non-controllable disturbance. The foundation of such model sits on two assumptions. First, each instruction type has a reference power consumption that slightly changes because of the processed data. Second, the action of sampling on a too short time epoch makes it impossible to predict a stable mix of executed instructions and thus the corresponding average power consumption.

$$P_k = d_k + \alpha \times A_{k-1} \tag{1}$$

The α parameter in Eq. 1 models the relationship between the controllable input and the power consumption. The model belongs to the family of deterministic auto-regressive ones with exogenous input (ARX) [28]. Starting from the methodology proposed in [1], we estimate α equal to 0.8. The low-pass filter is used to smooth the power consumption estimates that, from the experimental data at 2μs timing resolution, was found to contain high frequency components. Our experimental analysis suggested us to select p_1 equal to 0.5. We did not consider detailed models for the actuators. In fact, the experimental evaluation of the designed DFS and DCG actuators, would have led to model them as a pure delay of 80 ns and 300 ns, respectively. In this work we are not considering such delay in the stability analysis, since it is between 6 times and one order of magnitude smaller than the timing resolution of the system (2μs), and also the experimental results confirm the overall system stability. The delay model should instead be considered in the case of a slower actuator. In such a scenario, the impact on the phase margin can in fact void the system stability. The complete microarchitecture of the considered actuators are reported in Section 4.

Considering the detailed model of each component, the PID of the local controller has been finalized with an integral part, i.e., $p_0 = 1$, to ensure no error at steady-state and a derivative contribution to ensure enough responsiveness to power variations ($z_0 = 0.25$). Finally, we employ the root locus method to assign k_1 equal to 1, while ensuring the asymptotic stability of the closed-loop for both the employed actuators (DFS, DCG).

3.2 Global controller design

The global controller implements a single-input single-output (SISO) scheme that, for each time epoch k, takes the global power set point ($P_k^{Tot,SP}$) and outputs the available power budget ($P_k^{TotCorr,SP}$). Such global power set point ($P_k^{TotCorr,SP}$) is obtained by adding the total consumed power (P_k^{Tot}) and the correction factor S_k^{Tot} generated by the global controller. The correction factor S_k^{Tot} is related to the difference between the global set point $P_k^{Tot,SP}$ and the total consumed power. We note that the $P_k^{Tot,SP}$ quantity represents the average power consumption corresponding to the assigned energy budget imposed by the operating system (OS) or by the resource manager. The strategy adopted by an OS or the strategy used by a resource manager to decide the energy budget, falls outside the scope of this work. Last, the overall budget ($P_k^{TotCorr,SP}$) is split into the local set-points (see the *SIMO* in Figure 2).

Controller design - The global controller structure is meant to apply a correction factor to the total budget, i.e., $P_k^{Tot,SP}$. In particular the global controller consists in an integral component ($p_3 = 1$) that is fed with the error signal (e_k^{Tot}), i.e., the difference between the total power budget ($P_k^{Tot,SP}$) and the actual total consumed power (P_k^{Tot}). The controller works as an energy buffer to make available the power that hasn't been used so far in the next epochs. Similarly, the integral controller operates in the opposite direction if the total consumed power is higher than the set point. The correction factor S_k^{Tot} is algebraically added to the total budget $P_k^{Tot,SP}$ to get the total corrected power budget, i.e., $P_k^{TotCorr,SP}$. Such scheme allows to maximize the use of the energy budget in the long run by increasing or by reducing the total energy budget available to the computing units at each epoch k, depending on the energy spent in the past. Without lack of generality, the correction factor S_k^{Tot} is limited between $+/-2$ watts, while the overall budget $P_k^{TotCorr,SP}$ stays between 0 and 4 watts. Such numbers have been selected starting from the observed maximum power consumption of the reference platform, i.e., 1.6 W, to maximize the benefit of the proposed scheme and to size the width of buses and wires in the implemented microarchitecture.

We set k_0 equal to 0.01 to enforce a slow dynamics of the integral part of the controller. In this perspective, the pure integrator ensures that the global controller matches the imposed energy budget in the long run. We designed the controller to ensures the correction factor has a "slow" dynamics, namely milliseconds (see Figure 4), compared to the response time of the local controllers, $30\mu s$ (see Figure 3). The reason for this is twofold. First, the dynamics of the set-point in the global controller is expected to change in the order of seconds and above, thus a fast controller is not required. Second and more important, the difference between the dynamics of the global and of the local controllers, allows the latter to converge between two consecutive changes in the local set-points operated by the former.

The budget-split module ($SIMO$) distributes the total power budget to each computing element of the multi-core, by modifying the set point of each local controller. Such

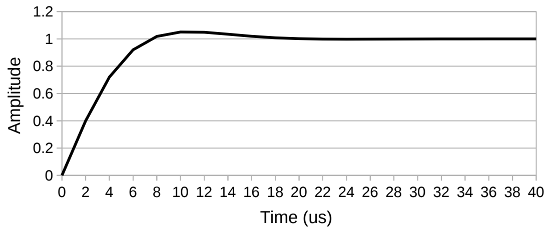


Fig. 3: Simulation of the step response of the local control-loop i-th.

distribution of the budget is achieved by means of a vector of θ values defined as follows:

$$\theta_i \in \mathbf{R}, \qquad \theta_i \geq 0, \qquad \sum_{i=1}^{nCpu} \theta_i = 1 \tag{2}$$

where $nCpu$ is the already defined number of computing units in the multi-core. The vector of θ values allows to distribute the total budget to the local controllers and it represents a flexible point to design the energy-allocation policies. We constrained the design of such policies to the supervisor algorithm that is discussed in Section 3.3.

Stability analysis - Eq. 3 defines the asymptotically stable closed-loop transfer function CL_{loc} of a single local controller between $P_{k,i}$ and $P_{k,i}^{SP}$.

$$CL_{loc} = \frac{P_{k,i}}{P_{k,i}^{SP}} = \frac{0.40 \times z - 0.12}{z^2 - 1.10 \times z - 0.38} \tag{3}$$

We leverage Eq. 3 to compute the transfer function of the equivalent system with input S_k^{Tot} and output P_k^{Tot} (see *Inner control-loop* in Figure 2). We note that it is a diagonal matrix of size $nCpu$ with all the entries of the main diagonal equal to the CL_{loc} transfer function. Such matrix is firstly left multiplied by the θ vector and, then, right multiplied by the ones vector, i.e., the MISO block in Figure 2. We note that $\sum_{i=1}^{nCpu} \theta_i = 1$ and the MISO is a vector of ones, thus the result of such matrix multiplication is the CL_{loc}. In summary, the corresponding transfer function of the closed inner control-loop is defined as follows:

$$CL_{inner} = \frac{P_k^{Tot}}{S_k^{Tot}} = \frac{CL_{loc}}{1 - CL_{loc}} \tag{4}$$

Starting from the closed-loop transfer function of the *Inner control-loop* (see Eq. 4), Eq. 5 defines the closed-loop transfer function of the *Global control-loop*.

$$CL_{glob} = \frac{P_k^{Tot}}{P_k^{Tot,SP}} = \frac{0.01 \times z - 0.01}{z^3 - 2.10 \times z^2 + 1.52 \times z - 0.39} \tag{5}$$

It is important to point out that Eq. 5 stays independent from the θ values. Such feature is needed to guarantee the asymptotic stability of the system, regardless of the θ coefficients, namely for any policy implemented in the Supervisor algorithm.

We implemented Eq 5 in Matlab 2018a to confirm its asymptotic stability. Moreover, Figure 4 reports a simulation of the transfer function defined in Eq. 5 varying the global set point. Results are reported considering different k_0 values around our selected working point, i.e., $k_0 = 0.01$.

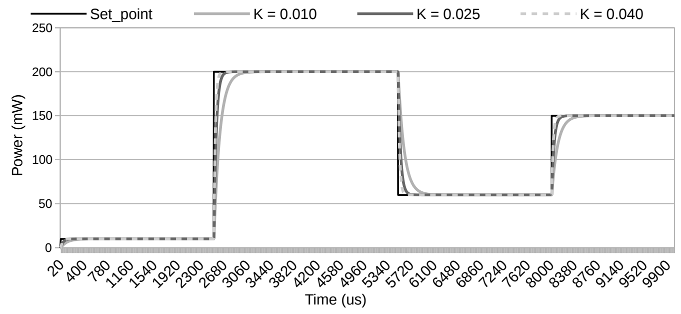


Fig. 4: Simulation of different step responses considering the global closed-loop transfer function. Results are reported considering different k_0 values.

The overall system transient time is in the order of one millisecond, while the zero-error at steady state is enforced by the pure integral part of the global controller (see $p_2 = 1$ in Figure 2). We note once more that the update of the global budget set-point ($P_k^{Tot,SP}$) is expected to show (at least) a dynamics in the order of hundreds of milliseconds. This is indeed driven by the operating system, that clearly justifies the reported transient time of the entire system.

3.3 Supervisor algorithm design

The supervisor algorithm allocates the total budget between the local controllers by periodically adjusting the θ values. From a mathematical perspective, the possibility of changing the θ values at run-time turns the proposed control scheme into a *switched system* that is made of a set of asymptotically stable linear systems. In particular, each linear system of such switched system is the hierarchical control scheme made of the global and all the local controllers with a specific instance of the θ vector of weights.

Exponential stability and dwell time - We note that the obtained switched system is a particular non-linear system made of a set of time-invariant asymptotically stable linear systems for which the stability condition has been proven to be fairly easy to obtain. In particular, we are interested in the exponential stability of the switched system [29] that is the specific asymptotic stability property required for non-linear systems. We leverage the work in [29] which demonstrates that any switched system made of a set of time invariant asymptotically stable linear systems, always has a minimum *dwell time* that ensures the exponential stability of the entire control scheme. The *dwell time* represents the minimum time interval between two consecutive changes of the θ values, and it guarantees the exponential stability of the switched system. Such time interval depends on the characteristics of the set of closed-loop systems making the switched system. It represents a single and easy to achieve condition to preserve the stability of the proposed control scheme. In particular, the work in [29] states that the system remains exponentially stable if the change of the θ values (switch signal) is “slow enough”. Starting from the closed-loop analysis discussed in Section 3.2 and Section 3.1, for the global and all the local controllers, and in the light of the findings in [29], we found that a dwell time equal to 64 microseconds satisfies the exponential stability condition. In particular, we are preventing any change of the θ values in the first 2 ms after a change in the global set point. Such restriction is not critical given the slow dynamics of the global set-point, i.e., hundreds of milliseconds at least. To this extent, we only have to guarantee that the dwell time can accommodate the transient time of the local controller, i.e., a dwell time bigger than 32 μs.

Supervisor algorithm - The *dwell time* represents the only constraint to the design of the supervisor algorithm. Apart from this easy-to-satisfy requirement, the supervisor is no longer involved in the stability analysis. To this extent, our coordinated scheme allows to define any energy-allocation policy, while still preserving the stability of the entire system.

We note that such result represents a critical advance in the design of the coordinated energy-budget and energy allocation schemes, since we provide a framework to virtually design any possible policy. In the rest of this Section, we describe an energy-allocation policy to balance the performance of the CPUs. Such policy is mainly intended to show the wide range of design possibilities offered by our control scheme, although it also achieves remarkable results with respect to the imposed performance goal.

We define a core-wise utility metric as the proxy for the performance. In particular, Eq. 6 defines the calculated utility ($uCalc_{i,k}$) of core i-th at time epoch k as the number of committed instructions (*wbInstr*) within a time epoch of 2μs corresponding to 100 clock cycles in our implementation running at 50MHz.

$$uCalc_{i,k} = \sum_{j=1}^{instrType} wbInstr_{i,k} * numCcInstrType_j$$
$$i \in \{\ 1,\ ..,\ nCpu\ \} \tag{6}$$

We weight each instruction according to its theoretical latency expressed in terms of clock cycles to complete its execution (*numCcInstrType*). This allows to acknowledge that executing fewer multi-cycle instructions rather than multiple single-cycle leads to the same utility. In particular, we use three instruction classes, i.e., *instrType*. Load/store instructions have a weight of 8, FPU ones have a weight of 16 while all the others, including the ALU ones, have a weight of 1. We note that, by construction, it is impossible to define an accurate utility model since some types of instructions, e.g., loads, stores, are affected by a non-deterministic latency dependent on the shared resource contention. Even single clock latency instructions may take multiple cycles to complete, due to data and/or structural hazards in the CPU. However, the experimental results show that the control-inspired nature of the supervisor controller makes it adaptive with respect to such possible model inaccuracies. We also note that the specific instance of the policy does not represent the key contribution of this work and thus any other utility metric and policy can be implemented to optimize different run-time goals. The calculated utility $uCalc_{i,k}$ is employed to define the CPU utility that represents the performance proxy for each core (see Eq 7). In particular, the supervisor algorithm updates the utility of the core i at the time epoch k ($u_{i,k}$) considering a weighted sum of the current utility and the calculated utility $uCalc_{i,k}$ as defined in Eq. 6 .

Algorithm 1 Pseudocode of Supervisor algorithm to enforce performance fairness through budget reallocation. The saturation controls on θ and on utility values, i.e., $\theta_k[i], u_k[i], uMean_k$, are not reported explicitly ($0 < \theta_k[i] < 1$, $0 < u_k[i] < 100$). The DWELL_TIME represents the time between two consecutive updates of the θ values and it is expressed as a number of clock cycles.

```
// signal if the CPU state is idle for each CPU
**input** $isCpuIdle[nCpu]$;
// set of committed instructions, for each CPU
**input** $wbInst[nCpu][InstrType]$;
// wait time before next update of the $\theta$ values
**input** $DWELL\_CYCLES$;
// control action for each core i-th
**input** $a[i]$;

**Program:**
$beginProgram$
  $state = IDLE$;   $\theta[i] = \frac{1}{|nCpu|}$,   $u[i] = 0 \quad \forall i$;

  **while**(true)
   **case**(state)

      **IDLE:**
       $begin$
          $\theta_{SlackIdle} \leftarrow 0$;   $uMean \leftarrow 0$;   $cnt \leftarrow 0$
          $set_{idle} \leftarrow \{\}$;   $set_{bal} \leftarrow \{\}$;   $set_{unbal} \leftarrow \{\}$;
          if($valid_i$)
             $state \leftarrow$ CALC_CORE_UTILITY;
       $end$

      **CHECK_DWELLTIME:**
       $begin$ // update core utility
          for(i ∈ { 1, .. , nCpu } )
             u[i] ← 0.5 * u[i] + 0.5 * $uCalc_i$;
          $cnt \leftarrow cnt + 1$;
          if (cnt==DWELL_CYCLES)
          begin
             $cnt \leftarrow 0$;
             $state \leftarrow$ PARTITION_CORES_IN_SETS;
          end
          else
             $state \leftarrow$ IDLE;
      $end$

      **STAGE_1:**
       $begin$ // partition cores in sets
          $state \leftarrow$ BALANCE_UTILITY;
          for(i ∈ { 1, .. , nCpu } )
          $begin$
             $if(isCpuIdle[i])$
             $begin$
                $\theta_{SlackIdle} \leftarrow \theta_{SlackIdle} + \theta[i]$
                $set_{idle} \leftarrow i$;//idle core
             $end$
             $elseif(isCpuIdle[i] == 0 \wedge \theta[i] == 0)$
                $set_{unbal} \leftarrow i$;//starting core
             $elseif(u[i] < calcMean(u) \wedge a[i] == 0)$
                $set_{bal} \leftarrow i$;//balanced cores
             $else$
             $begin$
                $set_{unbal} \leftarrow i$;//unbalanced cores
                $uMean_{unbal} \leftarrow uMean_{unbal} + u[i]$;
             $end$
          $end$

      **STAGE_2:**
       $begin$ // balance utlity
          $state \leftarrow$ ASSIGN_EXTRA_UTILITY;
          for(i ∈ $set_{unbal}$)
             $\theta[i] \leftarrow \theta[i] + (\frac{uMean_{unbal}}{|set_{unbal}|} - u[i])/100$
          for(i ∈ $set_{idle}$)
             $\theta[i] \leftarrow 0$
          for(i ∈ $set_{bal}$)
             $\theta[i] \leftarrow \theta[i]$

      **STAGE_3:**
       $begin$ //assign extra utility from idle cores.
          $state \leftarrow$ IDLE;
          for(i ∈ $set_{unbal}$)
             $\theta[i] \leftarrow \theta[i] + \frac{\theta_{SlackIdle}}{|set_{unbal}|}$
    **endcase**
 **endProgram**
```

$$u_{i,k} = 0.5 \times u_{i,k-1} + 0.5 \times uCalc_{i,k} \tag{7}$$

Starting from the defined utility ($u_{i,k}$), the supervisor implements a finite state machine (FSM) whose corresponding semantics is pointed out in Algorithm 1. The algorithm aims at optimizing the performance fairness between the cores, i.e., to maximize the balanced utility ($u_{i,k}$), by acting on the θ values to modify the energy-budget allocation to each local controller. Note that a CPU can have its utility lower than the average one due to either its energy-budget or the application behavior. In the former scenario, we can increase the application utility by increasing its local budget, i.e., by rising the set point of the corresponding local controller. In the latter scenario, the application is self-constraining its own utility and no increase of the energy budget can improve it. For example, an idle core always shows zero utility and a core executing a self-suspended application reports a close to zero utility.

To this extent, the supervisor algorithm identifies three sets to classify each core: *idle* (set_{idle}), *balanced* (set_{bal}) and *unbalanced* (set_{unbal}). The *idle* set contains all the idle cores. The *balanced* set contains all the cores running a self-limiting application for which no further action of the controller can increase its utility. The *unbalanced* set contains all the other cores.

The supervisor algorithm aims at maximizing the average utility of the *unbalanced* set with minimum standard deviation. This set is in fact the only set of cores that can benefit from a reshape of the θ values. Such goal allows to balance the utility between all the cores when the energy budget is low enough to actually constrain the execution of the applications and no core is idle. Moreover, the algorithm still optimizes the average utility of those cores that are running full-speed when idle cores and self-constrained applications are present.

The algorithm periodically updates the θ vector at *dwell time* resolution. However, as the rest of the proposed control scheme, it actually executes at a finer time resolution (2μs), in order to update the utility of each core at each time epoch k (see Eq 7 and lines 27-28 in Algorithm 1).

At the end of the period imposed by the *dwell time*, the algorithm starts by computing the next θ vector in three steps. First, each core is assigned to one of the three defined sets, i.e., idle, balanced and unbalanced (see *STAGE_1* in Algorithm 1). Moreover, it computes the sum of the θ values currently assigned to the cores in the *idle* set (see line 46 in Algorithm 1). Such quantity, i.e., $\theta_{SlackIdle}$ in Algorithm 1, will be later distributed between the unbalanced cores (see lines 73-74 in Algorithm 1) since the idle ones will not use it. We note that, apart from idle cores for which a signal of the architecture is available as input to the algorithm, the balanced cores are those for which the control action is zero and their utility is below the average utility computed on all the implemented cores (see line 50 in Algorithm 1). The second stage assigns the new θ to each core (see *STAGE_2* in Algorithm 1). The cores in the *idle* set obtain a θ value equal to zero. The cores in the *balanced* set maintain their previous value, while the ones in the *unbalanced* set have their θ corrected of a quantity equal to the algebraic difference between their utility and the average utility of the set (see line 64 in Algorithm 1). Such difference is downscaled by two orders of magnitude to match the range of values for each θ element, i.e., $0 \leq \theta \leq 1$ and $0 \leq u \leq 100$. The last stage evenly distributes the extra θ fraction taken from idle cores to the CPUs in the *unbalanced* set to optimize the overall system utilization (see *STAGE_3* in Algorithm 1).

4 Experimental Evaluation

This section reports the assessment of the proposed coordinated energy-budget and energy-allocation control scheme considering quad- and eight-core Single Instruction Multiple Data (SIMD) RISC processors [1] as reference computing platforms. The goal of this section is threefold. First, we experimentally assess the theoretical properties and the quality of the proposed scheme with respect to the imposed energy-budget and energy-allocation constraints. Second, we discuss the integration of OS and application level requirements into the proposed control scheme. Last, to demonstrate the feasibility of our proposal, the complete microarchitecture of the solution and its implementation on the commercial Nexys4-DDR board are presented. The rest of this section is organized in four parts. Section 4.1 presents the reference architecture as well as the microarchitecture of the proposed control scheme considering area, timing and performance design aspects. Section 4.2 discusses the quality metrics employed to assess the validity of the proposed control scheme. Section 4.3 points out the experimental results collected from a wide variety of use-case scenarios. Finally, Section 4.4 addresses a representative dynamic scenario where the global energy budget is changed by the OS, and the applications are imposing specific constraints on the energy-allocation policy.

4.1 Microarchitectural design

The entire design is synthesized, implemented and simulated at 50MHz targeting the Digilent Nexys4-DDR board [30] equipped with a Xilinx Artix-7 100t FPGA chip [31].

Reference architecture - We employ the *nu+* SIMD processor as the reference computing platform for which the complete description is available in [6]. To show the scalability, we employ both the quad- and the eight-core versions of the processor, for which each *nu+* core supports 16-way SIMD instructions but it is limited to a single-thread of execution. Multi-threaded applications run by using one core for each running thread. Such architecture can stress each part of the proposed control scheme. We used 20 programs from the WCET benchmark suite as representative applications [6]. The applications are running as bare-metal programs and we implemented the required run-time software to support multi-threaded execution, hence no OS support is offered and, thus, no GNU Linux libraries can be used. Such choice is meant to favor the implementation of the complete prototype of the reference computing platform, with emphasis to the hardware implementation of the proposed controller, i.e., the relevant contribution of this work. WCET can run on bare metal, but since they are indeed complete applications, they can stress each part of the computing platform.

Actuators - To argue the feasibility of the proposed solution, we are constrained by the characteristics of existing FPGA platforms, that offer DFS support only. In fact, due to the complexity of designing a dynamically scalable power rail for the reconfigurable logic, no commercial Xilinx FPGA integrates DVFS actuators. However, our controller can be coupled with any actuator and, to this purpose, we show the use of DFS and DCG actuators which have been both implemented on the considered Digilent board. The Dynamic Clock Gating (DCG) has been integrated to control each *nu+* core of the considered platform. It acts within each time epoch k by disabling the clock signal for a fraction of the time epoch to reduce the activity, and thus the power consumption, of the corresponding core. At the chosen time resolution, i.e., 2μs, the operating frequency of 50MHz shapes time epochs made of 100 clock cycles each. In particular, the designed DCG actuator leverages the programmable FPGA resource only, and takes up to 4 cycles to stop the clock cycles of the controlled CPU [1]. Differently, the Dynamic Frequency Scaling (DFS) actuator leverages the Xilinx Mixed Mode Clock Manager (MMCM) resources of the Artix-7 FPGA family [32]. The MMCM generates the clock frequency and it offers a reconfiguration port to change the operating frequency at run-time. However, the output clock signal is invalid during the reconfiguration period, thus preventing the use of a single MMCM to implement a proper DFS actuator. The work in [1] proposed a DFS actuator design employing two MMCM objects to synthesize a complete DFS actuator that always has a valid output clock signal, i.e., at most one MMCM is reconfiguring itself.

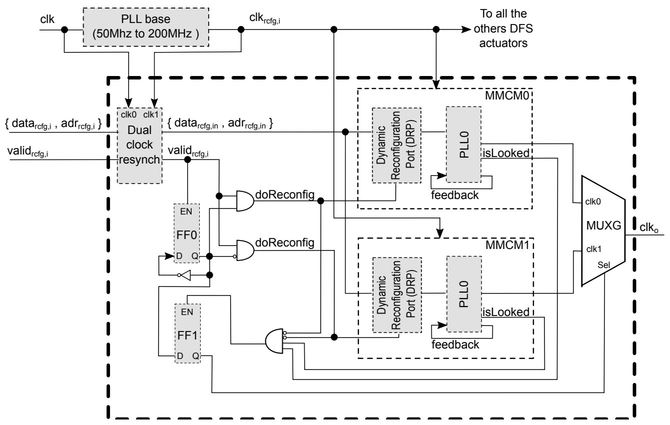

clk
PLL base (50Mhz to 200MHz)
clk$_{rcfg,i}$
To all the others DFS actuators
{ data$_{rcfg,i}$, adr$_{rcfg,i}$ }
valid$_{rcfg,i}$
clk0 clk1
Dual clock resynch
{ data$_{rcfg,in}$, adr$_{rcfg,in}$ }
valid$_{rcfg,i}$
MMCM0
Dynamic Reconfiguration Port (DRP)
PLL0
isLooked
feedback
doReconfig
doReconfig
EN
FF0
D Q
MMCM1
Dynamic Reconfiguration Port (DRP)
PLL0
isLooked
feedback
EN
FF1
D Q
clk0
MUXG
clk1
Sel
clk$_o$

Fig. 5: Proposed Dynamic Frequency Scaling (DFS) implementation for the Xilixn Artix-7 FPGA family. The DFS latency has been optimized to change the frequency in less than 300 ns.

The Xilinx *MUXG* is a glitch free multiplexer that ensures a correct transition between two clock frequencies. We note that the design, although it is robust, takes 1.2μs to lock to the requested operating frequency. Such delay directly impacts the stability analysis of the proposed control scheme, degrading the overall controller performance. To tackle such issues, we revisited the work in [1] to deliver a faster DFS actuator (see the new DFS architecture in Figure 5). Each MMCM implements a reconfiguration port, made of a data/address and of a validity signal. This triggers the synthesis of a new clock frequency (see $data_{rcfg}$, adr_{rcfg} and $valid_{rcfg}$ in Figure 5). We note that the MMCM takes 60 clock cycles to lock to the new frequency. Such number of cycles is measured with respect to the reconfiguration clock signal, i.e., clk_{rcfg}, and it remains constant across a wide range of frequencies of the reconfiguration clock signal. We optimize the DFS delay by rising the clock signal of the reconfiguration port to 200MHz, achieving a frequency update in less than 300 ns, i.e., 60 clock cycles, each of a period 5 ns. We also resynchronized the reconfiguration port of the actuator using a set of FIFO resynchronizers (see *Dual clock resynch* in Figure 5) which allows different operating frequencies between the actuator and the controller. In particular, the clock signal of the entire design (clk) is fed into a basic PLL to generate the 200MHz signal to clock the MMCM modules (see $clk_{rcfg,in}$ in Figure 5). The two clocks are fed into a Xilinx dual clock resynchronization module to resynchronize the wires driving the reconfiguration data. A $valid_{rcfg,in}$ is used to signal when a new frequency request happens while $data_{rcfg,in}$ and $adr_{rcfg,in}$ signals deliver the actual frequency value and the address of the memory-mapped MMCM register to write on. The final output signal (clk_o) to clock the CPU is selected by means of the Xilinx MUXG glitch-free component that commutes only when the lock signal of the reconfiguring MMCM is set. In particular, the lock wire signals when the MMCM has locked to the new requested operating frequency. We employ one flip-flop ($FF0$) to store the value of the currently used MMCM and a second flip-flop ($FF1$) to change the selector signal of the $MUXG$ component at the end of each frequency change.

Control scheme - The proposed control scheme is made of three parts: *i)* a set of $nCpu$ local controllers and actuators, *ii)* one global controller, and *iii)* one supervisor algorithm. The communication between the three parts of the controller takes one clock cycle. Moreover, the global and the local controllers take 3 cycles each to compute $P_k^{TotCorr,SP}$ and $C_{k,i}$, respectively, while the supervisor takes 6 cycles. Experimental results demonstrate that such delays do not affect the stability nor the effectiveness of the proposed control scheme. We note that the impact of the global and of the local controllers on the actuation also accounting for the communication is less that 10 clock cycles, i.e., the measured time for the architecture to execute 4 instructions, on average. The power values and the relative intermediate values, i.e., $P_k^{Tot,SP}$, e_k^{Tot}, P_k^{Tot}, S_k^{Tot}, $P_k^{TotCorr,SP}$, $P_{k,i}^{TotCorr,SP}$, $e_{k,i}$, $P_{k,i}$ and $P_{k,i}^{unfiltered}$, are expressed in milliwatts and are encoded using 13-bit signals. Such width is selected starting from the experimentally observed maximum power consumption which, for a single CPU, is less than 400mW. We note that the global controller limits the integral action between +/-2 watts to avoid the overflow of the $P_k^{TotCorr,SP}$ signal (see the saturation block in the *Global controller* in Figure 2). The correction factor encodes the sign in the most significant bit and its value in the remaining 12 bits. In particular, we observe a maximum power consumption of the eight-core platform (P_k^{Tot}) limited to 3W. This value, added to the maximum correction value S_k^{Tot} (2W), always remains within the maximum value encoded using 13 bits, namely 5W (see the saturation block within the *SIMO* block in Figure 2).

The remaining signals are encoded to fit their upper bounds. In particular, $C_{k,i}$ is an 8-bit signal that allows actuation values to be between 0% and 100%. $A_{k,i}$ is also encoded on 8 bits to express a number of idle cycles, if the DCG actuator is used, or a frequency between 100MHz and 1 MHz, if the DFS is used. Last, the vector that for each time epoch k and core i contains the number of committed instructions for each type, ($wbInstr_{k,i}[nCpu][InstrType]$) encodes each element using 8 bits. We note that the width of both the $A_{k,i}$ and of each element of the $wbInstr_{k,i}[nCpu][InstrType]$ vector, are sized according to the maximum value they can assume in 100 clock cycles, that is the selected time epoch period.

The update of each state variable and intermediate signal in the control scheme leverages shift operations in place of the expensive multiplications and divisions. In particular, we arithmetically left shift each value at the beginning of the computation to achieve such goal. We note that each multiplication and division is substituted by a set of base 2 multiplications and divisions, respectively. The latter approximate the required operation. For example, to divide a value num by 3 we approximate the result by means of a power of two divisions implemented as combinational logic. Their mathematical formulation is defined in Eq. 8.

$$\begin{aligned} res &= \frac{num}{4} + \frac{num}{16} + \frac{num}{64} \\ &= 0.25 \times num + 0.0625 \times num + 0.0156 \times num \\ &= 0.3281 \times num \end{aligned} \tag{8}$$

The design has been implemented targeting the Xilinx

TABLE 1: Area results for the proposed control scheme. For each controller and algorithm, the absolute number of used LUTs and FFs is reported.

Controller/Algorithm	LUTs	FFs
local (single)	275	138
global	463	276
supervisor	1816	275

FPGA flow. To this extent, Table 1 reports area results for each part of the proposed control scheme in terms of the absolute number of look-up-tables (LUTs) and flip-flops (FFs). Given the complexity of the proposed performance fairness policy, the supervisor algorithm dominates the area overhead in terms of LUTs, i.e., combinational logic elements, whilst the number of used flip-flops stays aligned with the other components. We also note the larger use of LUTs of the global controller compared to the local one. By Analyzing the results, we note that the gap does not depend on the complexity of the control policy implemented, rather on the need for merging (splitting) values (see *MISO* and *SIMO* in Figure 2) coming from (to) the local controllers. Such area also accounts for the logic required to wrap and to interface the actuators' building blocks, e.g., the MMCMs, with the rest of the control scheme. Considering that an Artix-7 100t chip features 126.8K Flip-Flops and 63.4K LUTs, the implementation of the proposed control scheme for for the quad-core reference platform reports a total resource utilization of 1103 FFs and 3379 LUTs, corresponding to a limited overhead of 0.86% and 5.3%, respectively. In the case of the eight-core, the overhead increases less than linearly with the number of cores and it is limited to 1.3% and 7.1% of the total FFs and LUTs of the platform. Moreover, the control scheme shows a positive timing slack of 5.1 ns when implemented at the selected operating frequency, i.e., 50 MHz. Even if such slack cannot tell the maximum operating frequency supported by the design, we are confident that achieving at least an operating frequency of 100 MHz is feasible.

4.2 Quality metrics

We define three metrics to measure the quality of the collected results. The global efficiency (EFF_g) expresses the efficiency of the assigned global power budget. The global overflow (OVF_g) measures the average overflow with respect to the imposed global set-point ($P_k^{Tot,SP}$). Last, the global utility (U_g) measures the obtained performance as intended in the supervisor algorithm, i.e., a fair balance of the utility within the set of *unbalanced* cores.

Global efficiency - The global efficiency (EFF_g), which is a percentage value between 0 and 100, measures the efficacy of the assigned global set-point to execute the computation. We develop the EFF_g figure of merit starting from the concept of the maximum consumed power at time epoch k for core i ($P_{k,i}^{max}$), which is defined as follows:

$$P_{k,i}^{max} = \frac{100}{100 - A_{k,i}} \times P_{k,i}, \quad 0 \leq A_{k,i} < 100 \tag{9}$$

In particular, Eq. 9 assumes 100 as the maximum number of uncontrolled clock cycles within any time-epoch k, while $A_{k,i}$ ($0 \leq A_{k,i} < 100$) is the control action that limits such value. We note that the control action is upper limited to 99 to prevent Eq. 9 from getting infinite values. In particular $A_{k,i}$ either stretches the clock cycle period (DFS) or selectively masks some positive edges of the clock to the computing logic (DCG).

Eq. 10 leverages the current power ($P_{k,i}$) and actuation ($A_{k,i}$) on the i-th core as well as its maximum power consumption ($P_{k,i}^{max}$) to define the power gap between the actual power and either the power max ($P_{k,i}^{max}$) or the current local set-point ($P_{k,i}^{SP}$). The $P_{k,i}^{cap}$ quantity accounts for the running applications that consume far less of the assigned power budget. We note that such applications must not contribute to reduce the effectiveness of the energy-budget scheme. In fact, the control action for these applications is null and it is not possible to increase neither the power consumption nor the performance.

$$P_{k,i}^{cap} = min\Big((P_{k,i}^{SP} - P_{k,i}), (P_{k,i}^{max} - P_{k,i})\Big) \tag{10}$$

Eq. 11 defines the energy-budget efficiency for the i-th CPU as the weighted average of the efficiency at each time epoch k. We already pointed out that the global set-point is equal to the sum of the local set-points. Thus, we define the total energy-budget efficiency (EFF_g) as the average of the local efficiencies, i.e., EFF_i, $\forall i \in nCpu$, (see Eq. 12).

$$EFF_i = \frac{1}{P_i^{SP}} \times \frac{\sum_{k=1}^{\#samples} max(0, P_{k,i}^{cap})}{\#samples} \tag{11}$$

$$EFF_g = \frac{\sum_{i=1}^{|nCpu|} EFF_i}{|nCpu|} \tag{12}$$

We note that the proposed definition of the global efficiency takes into account those scenarios for which the imposed global set-point is far bigger than the cumulative power consumption of all the applications running on the multi-core, e.g., when some cores are idle or when some applications are self-suspending. In particular, the efficiency is preserved in these scenarios.

Global overflow - While the global efficiency measures the quality of the assigned budget, the global overflow OVF_g measures the quality in fulfilling the total budget. In a nutshell, OVF_g measures the difference between the actual total power consumption and the set-point. Its value is limited to positive numbers and it is reported in terms of milliwatts.

To define such metric, we start by noting that the global controller implements an integral component that allows to employ an unused part of the budget at epoch k, in the following epochs. In the same way, an overuse of the budget at epoch k determines a negative correction (S_k^{Tot}) on the budget pertaining the subsequent epochs. Such implementation allows to constrain the consumed energy to the allocated energy budget in the long run. In this scenario it is worthless to measure the epoch-wise overflow, being the global setpoint dynamically corrected to optimize the use of the imposed energy budget. To acknowledge such

TABLE 2: Results considering the 4-core processor using the Dynamic Clock Gating (DCG) actuator. Results are reported in terms of efficiency (EFF_g), overflow (OVF) and utility ($U_{avg}^{set_{unbal}}$), considering different combinations of global set-points and number of running applications.

(a) EFF_g

Set Point	Running applications			
(mW)	1	2	3	4
100	99.78	97.53	95.69	93.94
200	100	99.92	98.81	94.14
300	100	100	99.99	96.94
400	100	100	100	99.85

(b) OVF_g

Set Point	Running applications			
(mW)	1	2	3	4
100	0.43	2.09	4.01	5.50
200	0	0.03	0.39	5.11
300	0	0	0	1.42
400	0	0	0	0.21

(c) $U_{avg}^{set_{unbal}}$

Set Point	Running applications			
(mW)	1	2	3	4
100	0	4.54	6.78	8.88
200	0	0.37	0.43	5.27
300	0	0	0.01	3.7
400	0	0	0	0.41

TABLE 3: Results considering the 4-core processor using the Dynamic Frequency Scaling (DFS) actuator. Results are reported in terms of efficiency (EFF_g), overflow (OVF) and utility ($U_{avg}^{set_{unbal}}$), considering different combinations of global set-points and number of running applications.

(a) EFF_g

Set Point	Running applications			
(mW)	1	2	3	4
100	100	98.04	92.25	94.4
200	100	99.99	96	93.54
300	100	100	99.99	98.85
400	100	100	100	99.23

(b) OVF_g

Set Point	Running applications			
(mW)	1	2	3	4
100	0.05	0.49	6.33	13.42
200	0	0.03	0.38	1.71
300	0	0.09	0.09	0.1
400	0	0.04	0.04	0.05

(c) $U_{avg}^{set_{unbal}}$

Set Point	Running applications			
(mW)	1	2	3	4
100	0.03	0.41	7.81	9.22
200	0	0.16	6.14	4.33
300	0	0.11	0.3	0.72
400	0	0.04	0.04	0.08

observation, we define the segmented overflow (OVF^{seg}) considering a set K of consecutive time-epochs, where the global set-point ($P_k^{Tot,SP}$) is fixed.

$$\begin{aligned} OVF^{seg} &= \frac{\sum_{i=1}^{|K|} P_i^{Tot}}{|K|} - P_i^{Tot,SP} \\ subject\ to \quad & P_i^{Tot,SP} = P_j^{Tot,SP} \quad \forall i,j \in K \end{aligned} \tag{13}$$

In particular, the OVF^{seg} quantity is defined as the algebraic difference between the averaged power consumption in the K set of time-epochs and the global set-point. Since the global set-point is fixed and the period of time within the set of epochs is expected to be sufficiently long, such quantity measures the overflow with respect to the imposed budget in the long run.

Starting from the OVF^{seg}, Eq. 14 defines the global overflow OVF_g as the average of the segmented overflow values to account for the changes of the global set-point. We limited OVF_g to positive values since we are interested in the overflow to the imposed set-point. The underflow would instead indicate that the set-point is too relaxed with respect to the actual power consumption.

$$\begin{aligned} OVF_g &= \frac{\sum_{i=1}^{|T|} OVF_i^{seg}}{|T|} \Bigg\rfloor_0 \\ subject\ to \quad & T = \{K_1, .., K_n\} \\ & K_z = \{k_1, .., k_m\}, \quad z \in \{1, .., n\} \\ & k_t\ is\ a\ time\ epoch, \quad t \in \{1, .., m\} \\ & P_{k_r}^{Tot,SP} = P_{k_q}^{Tot,SP} \quad \forall k_r, k_q \in K_z \end{aligned} \tag{14}$$

Global utility loss - The global utility loss ($ULoss_g$) measures the distance between the average utility and the utility of each core as the percentage relative error. To this extent, the smaller the better.

To define the global utility loss, we start by leveraging the concept of utility as our performance proxy to implement the fairness policy of the proposed control scheme (see the *Supervisor algorithm* in Section 3). In general, two applications showing the same power consumption can have different utility values. The utility is in fact related to the mix of instructions executed by each application and to the structural and data hazards.

As already discussed in Section 3, we enforce the fairness by minimizing the utility loss of the cores in the so-called unbalanced set, namely the cores that are neither idle nor have a utility lower than the average, and zero control action. In fact, idle CPUs show zero utility. Moreover, those CPUs for which the control action is zero cannot experience a utility improvement by increasing their set point, since their utility is limited by the application behavior itself. Thus, the unbalanced cores are those for which a change in the power set-point and the relative control-action affects the utility.

We leverage the utility definition of Eq. 15 that, for each time-epoch k, updates such quantity depending on the number of committed instructions each of them weighted according to its instruction type. For each time epoch k, the utility loss of the cores in the unbalanced set (U^{epoch}) is defined as follows:

$$ULoss^{epoch} = \frac{\sum_{i=1}^{|set_{unbal}|} \left| \frac{u_i - U_{avg}^{set_{unbal}}}{U_{avg}^{set_{unbal}}} \right|}{|set_{unbal}|} \times 100 \tag{15}$$

The quantity u_i is the utility of the i-th core at time k as defined in Eq. 15, while $U_{avg}^{set_{unbal}}$ is the average utility of the cores in the unbalanced set at time-epoch k (see lines 56 and 64 in Algorithm 1). To this extent, the utility loss at time-epoch k measures the distance, as the relative error, between the average utility and the utility of each core in the unbalanced set. We leverage the quantity $ULoss^{epoch}$ to define the global utility loss (U_g) as the $ULoss^{epoch}$ averaged on the set of the considered time-epochs (see Eq. 16).

$$\begin{aligned} ULoss_g &= \frac{\sum_{i=1}^{|K|} U_i^{epoch}}{|K|} \\ subject\ to \quad & K = \{k_1, ..k_n\} \\ & k_i\ is\ a\ time\ epoch,\ i \in \{1, .., n\} \end{aligned} \tag{16}$$

4.3 Performance, energy and scalability

This section reports the results in terms of the global efficiency (EFF_g), global overflow (OVF_g) and global utility loss ($ULoss_g$) considering the quad- and the eight-core reference processors, with a two-fold objective. First, we assess the performance of the proposed control scheme, i.e., the effective use of the allocated global budget without overflowing it, while considering a performance constraint (balanced utility for the running applications in the considered experiments). Second, we assess the scalability of the proposed solution by considering two processors, i.e. up to 8 cores. To ensure a statistical significance, each reported result is obtained as the average of 30 simulations of the same usecase, for which different randomly chosen benchmarks have been selected. Table 2 and Table 3 report the obtained results for the quad-core processor considering the use of the Dynamic Clock Gating (DCG) or the Dynamic Frequency Scaling (DFS) actuator, respectively. Results for the eight-core processor considering DCG and DFS are reported in Table 4 and Table 5, respectively. For each actuator and processor the results have been collected exploring two different design space directions: *i)* global set-point values and *ii)* number of executing applications. The use of different global set-points stresses the quality of the control scheme with a limited or a severely constrained budget. Differently, changing the number of executing applications stresses the quality of the control scheme for realistic scenarios, where the platform is not required to show its full computing power. In particular, we observe an average power consumption of 500 mW and of 1000 mW when 4 and 8 applications are executing. To this extent, we analyzed the quality of the control scheme considering four set-points for each processor that roughly correspond to 100%, 75%, 50% and 25% of the total required power. In the same way, we considered scenarios where the platform is used at different fractions of its total computing capacity. We stressed the quad-core processor at 25% (1 application), at 50% (2 applications), at 75% (3 applications) and at 100% (4 applications). Similarly we exercised the eight-core considering scenarios where 1 (12.5%), 2 (25%), 4 (50%), or 8 (100%) applications have been executed.

Regardless of the employed actuator and the processor (quad- or eight-core architecture), all the three considered metrics follow the same trend. In particular, they degrade with the reduction of the budget and with the increase in the executing applications. This depends on the severity of the scenario. If the controller does not drive the actuation, instead, there is no penalty from a metrics' perspective. Moreover, the use of 4- and 8-core processors show the scalability of the proposed controller since the increase in the number of cores does not degrade the considered quality metrics.

Global efficiency - Considering the 4-core processor, the average efficiency is 98.27% with a minimum value of 92% and 93.94% using DFS and DCG, respectively. The worst case values are obtained when the set-point is limited to 100 mW and the required platform computing capacity is above 75%, i.e., 3 or 4 applications are running. Such scenario occurs because each application is most likely blocked by the control scheme that is limiting its performance due to the imposed set-point. The high scalability of the proposed scheme is testified by the performance of the 8-core processor. In particular, the average global efficiency is 95.09% and 97.23% using the DCG and DFS, respectively.

Global overflow - Considering the global overflow on the 4-core processor, i.e., OVF_g, we note that the proposed control scheme is meant to align the total power consumption to the global set-point rather than keeping the former below the latter. In fact enforcing the latter condition penalizes the efficiency, although with the proposed design we are trading efficiency and overflow. The global overflow degrades when the control scheme operates under severe energy budget constraints, i.e., multiple running applications and low set-point value (100mW for 4-cores and 200mW for 8-cores). Such degradation is independent from the type of employed actuator and from the number of employed cores. For example, the worst case OVG_g are 5.5 mW and 13.42 mW using DCG and DFS for the 4-core CPU and 7.77 mW and 1.71 mW considering 8-cores with DCG and DFS actuators.

Global utility loss - The average global utility loss ($ULoss_g$) for the quad-core processor is 1.84% with a worst value of 8.88% and 9.22% using the DCG and DFS, respectively. Such metric shows a trend that is similar to the one observed for which the utility loss increases with lower set-points and an higher number of applications. In fact, such scenario is more likely constraining the applications that, thus, fall into the unbalanced set. As expected, an higher number of elements in the unbalanced set, makes the balancing process more difficult, thus lowering slightly the utility metric. We acknowledge a slightly increase in the utility loss, i.e., 1% when the 8-core processor is considered. Such loss is due to the need of balancing a larger number of cores. As a second critical observation, the optimal value of the utility loss metric ($ULoss_g$) is determined by the supervisor algorithm that enforces the θ values. In particular, the θ values change in the order of 64 μs, thus showing a dynamic that is far slower than the latency of both the employed actuators. To this extent, even the DFS has the time to converge before the subsequent change of the θ values, thus showing results aligned to the one reported when the the DCG is used.

4.4 Dynamic scenario

This section discusses the results obtained from the execution of 4 applications on the 4-core processor employing the DCG actuator. It then considers the interaction between

TABLE 4: Results considering the 8-core processor using the Dynamic Clock Gating (DCG) actuator. Results are reported in terms of efficiency (EFF_g), overflow (OVF) and utility ($U_{avg}^{set_{unbal}}$), considering different combinations of global set-points and number of running applications.

(a) EFF_g

Set Point	Running applications			
(mW)	1	2	4	8
200	100	99.92	94.14	91.11
400	100	100	99.85	92,59
600	100	100	100	97.08
800	100	100	100	99.58

(b) OVF_g

Set Point	Running applications			
(mW)	1	2	4	8
200	0	0.03	5.11	7.77
400	0	0	0.21	6.09
600	0	0	0	2.24
800	0	0	0	0.18

(c) $U_{avg}^{set_{unbal}}$

Set Point	Running applications			
(mW)	1	2	4	8
200	0	0.37	5.27	8.51
400	0	0	0.41	6.09
600	0	0	0.17	4.16
800	0	0	0	0.77

TABLE 5: Results considering the 8-core processor using the Dynamic Frequency Scaling (DFS) actuator. Results are reported in terms of efficiency (EFF_g), overflow (OVF) and utility ($U_{avg}^{set_{unbal}}$), considering different combinations of global set-points and number of running applications.

(a) EFF_g

Set Point	Running applications			
(mW)	1	2	4	8
200	100	99.99	93.54	95.91
400	100	100	99.23	95.85
600	100	100	99.71	98.02
800	100	100	100	99.12

(b) OVF_g

Set Point	Running applications			
(mW)	1	2	4	8
200	0	0.03	6.33	1.71
400	0	0.04	0.38	0.05
600	0	0.04	0.09	0.03
800	0	0.09	0.04	0.03

(c) $U_{avg}^{set_{unbal}}$

Set Point	Running applications			
(mW)	1	2	4	8
200	0	0.16	4.33	9.49
400	0	0.04	0.08	5.88
600	0	0.03	0.05	3.17
800	0	0.03	0.04	1.01

the controller, the OS and the applications. The final goal is to demonstrate the possibility to seamlessly integrate standard OS-level and application-level policies to be effectively actuated by the proposed controller. For example, the OS can impose a different global energy-budget at run-time or limit the energy quota on specific applications [8]. Differently, an application that is self-monitoring through emerging self-adaptive distributed management schemes, e.g., MARGOT [8], might need to change its energy quota to optimize its own quality-of-service.

We note that the definition of the QoS for an application or that of the policy needed to specify the global energy-budget, are out of scope here. Our work aims instead at making it possible to account for such decisions and constraints into our control scheme. In particular, a self-monitoring application can regularly ask to change its θ value depending on its application-level policy. The critical aspect to note is that such application behavior, seamlessly integrates with the proposed controller that fixes the θ required by the application. At the same time it is ensured that the energy-allocation policy implemented in the supervisor continues to manage the remaining running cores.

Results are reported in Figure 6 considering the signals of the 4 local controllers (see Figure 6a- 6d) and the single global one (see Figure 6e). According to the notation defined in Figure 2 for the control scheme, for each local controller i-th we reported the power consumption ($P_{k,i}$), the power consumption set-point ($P_{k,i}^{SP}$), the actuation signal ($A_{k,i}$) as well as the utility ($Utility_i$) and the assigned θ value (Θ_i). Moreover, the total budget with slack ($P_k^{TotCorr,SP}$), the total consumed power (P_k^{Tot}), the slack (S_k^{Tot}) and the total power set-point ($P_k^{Tot,SP}$) are displayed for the global controller.

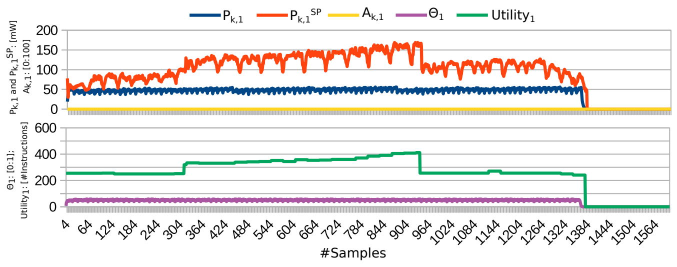
The proposed scenario allows to discuss three aspects. First, the OS is forcing a low θ_4 value between sample 310 and 364. Such action has two consequences. Local controller 4 starts actuating due to the reduced power set-point ($P_{k,i}$) for application 4 and thus we observe a reduction in the utility for the same application. In contrast, the remaining applications experience an increase in their utility, due to the extra θ fraction removed from application 4 that has been redistributed among the other running applications.

Second, the OS is reducing the global power budget at time sample 1264 with two distinct consequences. The total power set-point ($P_k^{TotCorr,SP}$) is gracefully lowering as a consequence of the energy budget buffer implemented in the global controller; in other words, $p_2 = 1$ generates an integrator in the global controller (see Figure 2). Such buffer ensures a graceful degradation of the performance of the running applications, while maintaining stable the difference between the sum of the global set-points and the consumed energy, i.e., the error (e_k^{Tot}) is 0 at steady state. In particular, the system is using the energy accumulated in the buffer until the reduction of the global set-point at time sample 1264. The energy in the buffer increases because the applications were not using all the available energy budget (see the increase of the S_k^{Tot} until sample 1264).

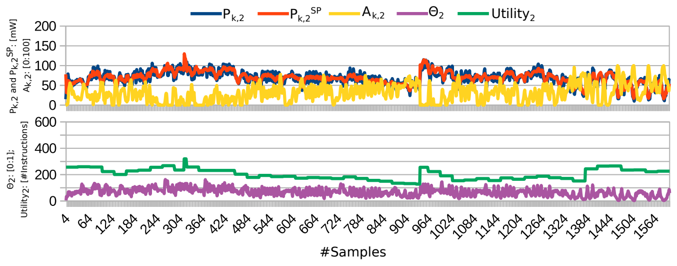
The last scenario is devoted to the evaluation of the system when application 1 terminates around sample 1384. The controller acknowledges the end of the application by removing its core from the set of unbalanced ones. To this extent, the energy budget allocated to the terminated application is distributed to the other three running applications, with a net increase in their utility. Such results demonstrate the adaptivity of the system to an external event.

5 Conclusions

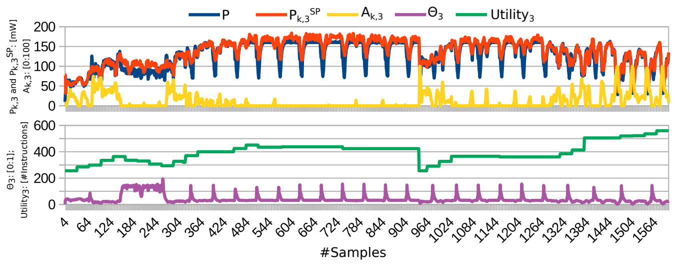
This work presented a coordinated control scheme to optimize the energy-budget and energy-allocation for multi-cores, also ensuring the exponential stability of the overall system. Our proposal is neither an ad-hoc heuristic nor an energy-budgeting algorithm, but a complete framework to design any energy allocation policy. It advances the



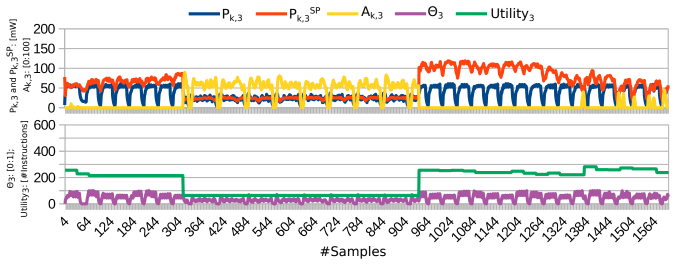
(a) Local controller 1



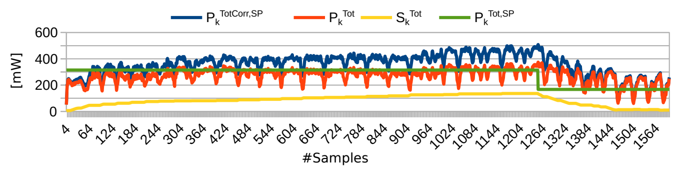
(b) Local controller 2



(c) Local controller 3



(d) Local controller 4



(e) Global Controller

Fig. 6: Evaluation over the time of the actuation and monitored signals for the global and the 4-local controllers considering a quad-core processor employing the DCG actuator. Application 4 imposes its own θ_4 value between sample 320 and 950 (see Figure 6d), application 1 terminates at sample 1390 (see Figure 6a) and OS imposes a new energy budget at sample 1264 (see Figure 6e).

state-of-the-art in three ways. First, it allows to cast any energy allocation policy as an algorithm of the supervisor controller while maintaining the global control scheme exponentially stable, regardless of the implemented policy. Second, the proposed controller can seamlessly integrate constraints from the OS and application level, also working with different actuators, still preserving the stability property. Third, the scalability of the proposed solution has been demonstrated through the use of a quad- and an eight-core processors as computing platforms. Moreover, the low-area overhead in the standard-logic implementation of the proposed controller, highlighted its flexibility and wide applicability.

We validated the proposed scheme on a real prototype considering a quad- and an eight-core processor as computing platforms as well as two actuators, i.e., DFS and DCG. In particular, the complete design has been synthesized, placed and routed on a Nexys4-DDR board featuring a Xilinx Artix-7 100t FPGA chip and considering a 50 MHz clock since it is the maximum operating frequency supported by the quad-core. The obtained area occupation is limited to 0.86% (FFs) and 5.3% (LUTs) of the FPGA chip considering the reference quad-core. We use three metrics to assess the quality of the proposed control scheme: the respect of the imposed energy-budget (OVF_g), the performance loss due to control scheme ($ULoss_g$) and the quality in terms of how efficiently the granted energy budget (EFF_g) is exploited. In particular, we collect results for a huge variety of realistic scenarios and the statistical significance has been considered by executing each scenario 30 times. The obtained results show valuable achievements: the average EFF_g is 98.27% (worst case 92.25%), the average OVF_g is 1.43 mW (worst case 13.42 mW) and the average $ULoss_g$ is 1.87% (worst case 9.22%).

References

[1] L. Cremona, W. Fornaciari, and D. Zoni, “All-digital energy-constrained controller for general-purpose accelerators and cpus,” *IEEE Embedded Systems Letters*, p. 4, Accepted for publication 2019.

[2] P. Chaparro, J. GonzÃąles, G. Magklis, Q. Cai, and A. GonzÃąlez, “Understanding the thermal implications of multi-core architectures,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 8, pp. 1055–1065, Aug 2007.

[3] D. Rodopoulos, F. Catthoor, and D. Soudris, “Tackling performance variability due to ras mechanisms with pid-controlled dvfs,” *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 156–159, July 2015.

[4] D. Zoni and W. Fornaciari, “Modeling dvfs and power-gating actuators for cycle-accurate noc-based simulators,” *J. Emerg. Technol. Comput. Syst.*, vol. 12, no. 3, pp. 27:1–27:24, Sep. 2015. [Online]. Available: http://doi.acm.org/10.1145/2751561

[5] M. Najem, P. Benoit, M. E. Ahmad, G. Sassatelli, and L. Torres, “A design-time method for building cost-effective run-time power monitoring,” *IEEE TCAD*, vol. 36, no. 7, pp. 1153–1166, July 2017.

[6] D. Zoni, L. Cremona, A. Cilardo, M. Gagliardi, and W. Fornaciari, “Powertap: All-digital power meter modeling for run-time power monitoring,” *MICPRO*, vol. 63, pp. 128 – 139, 2018.

[7] D. J. Pagliari, V. Peluso, Y. Chen, A. Calimera, E. Macii, and M. Poncino, “All-digital embedded meters for on-line power estimation,” in *DATE*, March 2018, pp. 743–748.

[8] D. Gadioli, E. Vitali, G. Palermo, and C. Silvano, “margot: A dynamic autotuning framework for self-aware approximate computing,” *IEEE Transactions on Computers*, vol. 68, no. 5, pp. 713–728, May 2019.

[9] P. Bellasi, G. Massari, and W. Fornaciari, “Effective runtime resource management using Linux control groups with the Barbeque RTRM framework,” *ACM TECS*, vol. 14, no. 2, pp. 1–17, 2015.

[10] R. Rodrigues, A. Annamalai, I. Koren, and S. Kundu, “A study on the use of performance counters to estimate power in microprocessors,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 60, no. 12, pp. 882–886, Dec 2013.

[11] A. Pathania, A. E. Irimiea, A. Prakash, and T. Mitra, "Power-performance modelling of mobile gaming workloads on heterogeneous MPSoCs," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2015, pp. 1–6.

[12] G. Singla, G. Kaur, A. K. Unver, and U. Y. Ogras, "Predictive dynamic thermal and power management for heterogeneous mobile platforms," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015, pp. 960–965.

[13] M. J. Walker, S. Diestelhorst, A. Hansson, A. K. Das, S. Yang, B. M. Al-Hashimi, and G. V. Merrett, "Accurate and stable run-time power modeling for mobile and embedded CPUs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 1, pp. 106–119, Jan 2017.

[14] M. Pricopi, T. S. Muthukaruppan, V. Venkataramani, T. Mitra, and S. Vishin, "Power-performance modeling on asymmetric multi-cores," in *Proceedings of the 2013 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, ser. CASES '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 15:1–15:10.

[15] R. Xu, D. Mossé, and R. Melhem, "Minimizing expected energy consumption in real-time systems through dynamic voltage scaling," *ACM Trans. Comput. Syst.*, vol. 25, no. 4, Dec. 2007.

[16] R. Begum, D. Werner, M. Hempstead, G. Prasad, and G. Challen, "Energy-performance trade-offs on energy-constrained devices with multi-component dvfs," in *2015 IEEE International Symposium on Workload Characterization*, Oct 2015, pp. 34–43.

[17] N. Bansal, T. Kimbrel, and K. Pruhs, "Speed scaling to manage energy and temperature," *J. ACM*, vol. 54, no. 1, pp. 3:1–3:39, Mar. 2007. [Online]. Available: http://doi.acm.org/10.1145/1206035.1206038

[18] T. R. da Rosa, V. Larrea, N. Calazans, and F. G. Moraes, "Power consumption reduction in mpsocs through dfs," in *SBCCI*, 2012.

[19] A. Sansottera, D. Zoni, P. Cremonesi, and W. Fornaciari, "Consolidation of multi-tier workloads with performance and reliability constraints," in *2012 International Conference on High Performance Computing Simulation (HPCS)*, July 2012, pp. 74–83.

[20] Y. Liu, G. Cox, Q. Deng, S. C. Draper, and R. Bianchini, "Fastcap: An efficient and fair algorithm for power capping in many-core systems," in *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, April 2016, pp. 57–68.

[21] M. Chen, X. Wang, and X. Li, "Coordinating processor and main memory for efficientserver power control," in *Proceedings of the International Conference on Supercomputing*, ser. ICS '11. New York, NY, USA: ACM, 2011, pp. 130–140.

[22] K. Ma, X. Li, M. Chen, and X. Wang, "Scalable power control for many-core architectures running multi-threaded applications," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ser. ISCA '11. New York, NY, USA: ACM, 2011, pp. 449–460.

[23] J. M. Cebrin, J. L. Aragon, and S. Kaxiras, "Power token balancing: Adapting cmps to power constraints for parallel multithreaded workloads," in *2011 IEEE International Parallel Distributed Processing Symposium*, May 2011, pp. 431–442.

[24] P. Bogdan, R. Marculescu, and S. Jain, "Dynamic power management for multidomain system-on-chip platforms: An optimal control approach," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 18, no. 4, pp. 46:1–46:20, Oct. 2013. [Online]. Available: http://doi.acm.org/10.1145/2504904

[25] P. Bogdan and R. Marculescu, "Statistical physics approaches for network-on-chip traffic characterization," in *Proceedings of the 7th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES+ISSS '09. New York, NY, USA: ACM, 2009, pp. 461–470. [Online]. Available: http://doi.acm.org/10.1145/1629435.1629498

[26] R. David, P. Bogdan, and R. Marculescu, "Dynamic power management for multicores: Case study using the intel scc," in *2012 IEEE/IFIP 20th International Conference on VLSI and System-on-Chip (VLSI-SoC)*, Oct 2012, pp. 147–152.

[29] J. P. Hespanha and A. S. Morse, "Stability of switched systems with average dwell-time," in *Proceedings of the 38th IEEE conference on decision and control (Cat. No. 99CH36304)*, vol. 3. IEEE, 1999, pp. 2655–2660.

[27] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan, "Power-management architecture of the intel microarchitecture code-named sandy bridge," *IEEE Micro*, vol. 32, no. 2, pp. 20–27, March 2012.

[28] L. Ljung, *System identification: theory for the user*. Prentice-Hall, 1987.

[30] Xilinx, "Digilent Nexys4-DDR," https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/start.

[31] ——, "7 Series FPGAs Data Sheet: Overview," https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf.

[32] ——, "7 Series FPGAs Clocking Resources," https://www.xilinx.com/support/documentation/user_guides/ug472_7Series_Clocking.pdf.

Davide Zoni is a Post-doc Researcher at Politecnico di Milano, Italy. He published more than 40 papers in journals and conference proceedings. His research interests include RTL design and optimization for multi-cores with particular emphasis on low power methodologies and hardware-level countermeasures to side-channel attacks. He received two HiPEAC collaboration grants in 2013 and 2014 and two HiPEAC industrial grant in 2015 and 2017. In 2019, he won the *Switch2Product - Innovation Challenge* competition (www.s2p.it). He is also the principal investigator of the *Lightweight Application-specific Modular Processor* (LAMP) platform (www.lamp-platform.org).

Luca Cremona received his B.S. in Engineering of Computer Systems in 2014 and his M.S. in Computer Science and Engineering in 2017, both from Politecnico di Milano. He is currently a PhD student in the same university and his research interests include RTL power modeling and management for multi-cores.

William Fornaciari is Associate Professor at Politecnico di Milano, Italy. He published 6 books and around 300 papers in int. journals and conferences, collecting 5 best paper awards and one certification of appreciation from IEEE. He holds three international patents on low power design. In 2019, he won the *Switch2Product - Innovation Challenge* competition (www.s2p.it). His research interests include embedded and cyber-physical systems, energy-aware design of sw and hw, run-time management of resources, design optimization and thermal management of multi-many cores and NoCs.