computers

MDPI

Article

Feedback-Based Admission Control for Firm Real-Time Task Allocation with Dynamic Voltage and Frequency Scaling

Piotr Dziurzanski [1,*] and Amit Kumar Singh [2]

[1] Department of Computer Science, University of York, York YO10 5GH, UK
[2] School of Computer Science and Electronic Engineering, University of Essex, Colchester CO4 3SQ, UK; a.k.singh@essex.ac.uk
* Correspondence: piotr.dziurzanski@york.ac.uk; Tel.: +44-01904-325547

Received: 13 March 2018; Accepted: 13 April 2018; Published: 16 April 2018

Abstract: Feedback-based mechanisms can be employed to monitor the performance of Multiprocessor Systems-on-Chips (MPSoCs) and steer the task execution even if the exact knowledge of the workload is unknown a priori. In particular, traditional proportional-integral controllers can be used with firm real-time tasks to either admit them to the processing cores or reject in order not to violate the timeliness of the already admitted tasks. During periods with a lower computational power demand, dynamic voltage and frequency scaling (DVFS) can be used to reduce the dissipation of energy in the cores while still not violating the tasks' time constraints. Depending on the workload pattern and weight, platform size and the granularity of DVFS, energy savings can reach even 60% at the cost of a slight performance degradation.

Keywords: feedback; admission control; firm real-time; DVFS; MPSoC; task allocation

1. Introduction

In the majority of current research on hard or firm real-time taskset scheduling in multi-core systems, an assumption is made that the taskset is known a priori [1]. As a result, it is possible to apply in advance the traditional schedulability tests to check if a specific taskset violates its timing constraints. However, numerous contemporary real-time systems execute dynamic workloads, where tasks join and leave the system in a way not known during the design time [2]. Dynamic tasksets are usually not consistent with relatively simple periodic or sporadic task models. However, it is rather difficult to find a schedulability analysis method for multi-core systems that is based on more sophisticated models [3]. The tasksets that are not compatible with these basic models, for example including asynchronous tasks, are more commonly analysed in the High Performance Computing (HPC) domain, for instance in [4]. The experience of the HPC community with such tasksets can help introduce the corresponding workload models in the real-time multi-core systems.

On HPC systems, tasks allocation and scheduling heuristics that employ feedback control have been demonstrated to be beneficial in the case of dynamic workloads as they can lead to increased platform utilisation and satisfy the timing constraints [5]. Despite many successful applications in the HPC domain, these heuristics have been, to our best knowledge, never used in multi-core embedded platforms with hard or firm real-time constraints.

According to the IEEE Technical Committee on Real-Time Systems, three real-time types can be singled out based on the consequences of missing a deadline: hard, where any deadline violation may jeopardise the correct behavior of the entire system, firm, where it does not endanger the correct system behaviour but the task late completion is worthless, and soft, where a late task has still a reduced value for the system. Control-theory-based mechanisms are relatively often applied to soft real-time systems [6], but this is not the case for the remaining real-time system types. For example, the Roadmap on Control of Real-Time Computing Systems [7], created during the Network of Excellence ARTIST2 program, explicitly states that feedback scheduling is not appropriate for applications with strict timing constraints, as algorithms in feedback-based systems use prior error values to perform actions relevant for the current time instant. However, applying feedback mechanisms to tasks with even hard real-time constraints is possible and beneficial especially in the situation when there is no a priori knowledge about the system load and its service performance [8]. For example, this approach has been applied to find a compromise between several objectives, such as the minimum slack or maximum utilisation of cores in [9]. It can be also employed as a heuristic method for approximated schedulability analysis for hard real-time tasks [10]. Similarly, feedback mechanisms can be used for decreasing energy dissipation. For example, in [11], controllers are used to find a trade-off between dissipated energy and task processing speed not violating any deadline by utilising Dynamic Voltage and Frequency Scaling (DVFS), one of the most popular power-saving techniques in digital circuits.

In modern processors, e.g., Intel Xeon (Santa Clara, CA, USA), typically, a limited range of discrete voltage/frequency levels is offered, following the Advanced Configuration and Power Interface (ACPI) open standard (http://www.uefi.org). This standard defines the operating state of a processor as C0, whereas halt, stop-clock and sleep states are referred to as C1, C2 and C3, respectively. Some chip vendors have introduced additional C-states as well, e.g., C6 in Intel Xeon named deep power conservation state [12]. In operating state C0, the core can be in one of the predefined power-performance states referred to as P-states. The P-states with higher indices are characterised by lower performance and lower energy dissipation.

To make a right decision about voltage scaling, assorted metrics provided by monitoring infrastructure tools and services can be used, for example the infrastructure utilisation or latency between the input and output timestamps [13]. In contemporary Multiprocessor Systems on Chips (MPSoCs), their cores can operate at various voltage and frequency levels in the same time instant, so selecting a core for a real-time task becomes a more sophisticated problem even for a homogeneous architecture [14]. It is caused by the fact that assigning a task to a lower-voltage core can lead to a deadline violation, whereas this deadline would be met after allocation to a core with a higher frequency. Such scheduling policies considering various voltage and frequency levels are referred to as voltage scheduling.

A task can be allocated to a core in an MPSoC in a static or dynamic manner. The former is advised in case the workload is known in advance, whereas dynamic allocation, performed after the release of a task, is the only possibility for workloads not known a priori [15]. Dynamic task mapping in MPSoCs supporting DVFS is even more challenging, as not only the target core is to be chosen, but also the range of the available voltage levels is to be considered. Important exemplifications of such allocation are the scheduling algorithms in Windows or Linux (since kernel version 2.6), which utilise dynamic frequency scaling for contemporary multicore processors from Intel (SpeedStep technology) and AMD (Santa Clara, CA, USA) (PowerNow! or Cool'n'Quiet technology). In modern operating systems, selecting a proper P-state is done by the governor. For example, the ondemand governor in Linux selects P0 instantly in the case of a high load, whereas the conservative governor changes the current P-state gradually [16]. The goal of these heuristics is to keep the processor utilisation close to 90% by reactive decreasing or increasing frequency with certain heuristics [17]. However, in the case of real-time tasks, the frequency should be chosen in a way not only to maintain a certain processor utilisation, but also meet the timing constraints. Hence, custom governors need to be used. Such governor is proposed in this paper.

Custom governors can operate on a per-core or per-chip (chip-wide) basis. An interesting comparison between per-chip and per-core DVFS is given in [18], according to which per-core DVFS lead to about 20% more energy savings than a typical chip-wide DVFS with off-chip regulators. Despite such results, per-core DVFS has not been implemented widely in hardware. For example, the active cores in contemporary Intel i7 processors have to work with the same frequency and voltage in steady states, whereas cores in AMD processors can operate with various frequencies, but still with a single voltage value, required by the core in the fastest P-state [19].

This paper concerns a dynamic taskset mapping using a custom governor algorithm for both per-chip and per-core DVFS. The proposed algorithm performs a dynamic allocation of firm real-time tasks to resources. Two-level resource allocation is performed. Firstly, a global dispatcher fetches a task from a common task queue and selects the least utilised multi-core processor. Then, a local admission control block performs an approximate, but fast task schedulability analysis and selects an appropriate voltage and frequency level for the processing core. The processing cores are switched to lower frequency states when the current workload demands lower computational power. Consequently, a considerable amount of energy can be saved without a significant impact on the performance.

A preliminary stage of the research presented in this paper has been published in [20]. However, in that publication, only one architecture platform has been considered, in which both voltage and frequency are scaled in a per chip manner. This paper differs in the following aspects: (i) two more platform architectures have been added and compared against the previously proposed one; (ii) a new scheduling algorithm, suitable for the newly proposed architectures, has been proposed; (iii) a detailed discussion of the experimental results comparing the newly proposed platform architecture with the previously proposed one is presented; and (iv) more realistic platform sizes are used for comparison.

This paper is organized into seven sections. Section 2 reviews related research work. In Section 3, the assumptions regarding the application and platform models are presented and the problem considered in this paper is formulated. In Section 4, some preliminary information regarding the elements of control theory applied to the proposed solution is provided. Section 5 describes the proposed algorithm for energy-efficient task admission control. Section 6 discusses the experiments and demonstrates the obtained results. Section 7 presents the final remarks.

2. Related Work

To map tasks to cores, techniques originated from control-theory offer soft real-time guarantees in which an infrequent deadline miss is possible. Therefore, they cannot be applied to time-critical systems [5], as hard real-time guarantees are required. Related to the firm or hard real-time systems, relatively little work exists. In these systems, the task dispatching should ensure admission control and guaranteed resource provisions. This implies that the execution of a task can start only when (i) the system can allocate a necessary resource budget to meet its timing requirements; and (ii) guarantee that no access of a task being executed to its allocated resources is denied or blocked by any other tasks [21]. The requirements of time critical systems, e.g., automotive and avionic systems, where meeting the timing constraints is essential [22,23], can be fulfilled by providing such kind of guaranteed facilitates.

For scheduling in hard or firm real-time systems, typically, the worst-case execution time (WCET) of each task in the workload should be known in advance in order to guarantee the schedulability of the whole system [1]. However, WCET and the average execution time (ET) can differ substantially as shown by a number of experimental results [24]. Therefore, system resources are often underutilised during execution. This provides an opportunity to use the resources differently than in the WCET scenario while executing tasks in a typical scenario.

For typical scenarios that are different from the worst case, a control mechanism to adjust the voltage level of unicore portable devices in a control-theory-based manner to conserve energy is proposed in [8]. In [25], proportional-integral-derivative (PID) controllers are proposed that determine voltage of systems dealing with the workload not accurately known in advance. With regards to dynamic workloads, the authors also interpreted the meaning of the discrete PID equation terms. To predict the future system load based on the workload rate of change, they proposed a heuristic that leads to significant reduction in energy consumption. The authors have also demonstrated that, with the changes in the PID controller parameters, the design space is not significantly changed.

However, in this work, the controller does not use any feedback information about the processing core status. Rather, it is used to predict the future workload.

A feedback-based approach is presented in [26], where tasks' slack time is managed to conserve energy in real-time systems. The real execution time of a task is predicted by a PID controller and is usually lower than its WCET. For a task, each execution time slot is split into two parts, where the first part is executed with a lower voltage assuming the controller's predicted execution time. In case the task does not finish in the first slot, the core is switched to its highest voltage to guarantee the execution completion before the deadline.

For soft real-time systems that allow a slight amount of deadline violations, energy-based feedback scheduler and a power-aware optimization algorithm is proposed in [27]. The load of the system is adjusted by controlling the speed of the task execution with the help of a proportional (P) controller used by the dispatcher. The appropriate speed for each task and the proper voltage level is computed by employing a greedy algorithm. For dispatching the tasks, earliest deadline first (EDF) or Rate-monotonic scheduling (RMS) are used. As the controlled variable, energy saving ratio over the sampling period is used and the controller's manipulated variable is considered as the worst-case utilisation.

In [28], voltage scheduling for MPSoCs has been initiated. The configuration of an MPSoC platform has to be set into a pipelined multi-layer way, where feedback policies for adjusted core frequencies of processors are based on the occupancy levels of the task input queues of each processor. The authors have implemented and compared linear and nonlinear feedback strategies. However, for core configurations other than the pipelined ones, the scheme is not applicable.

A formal analytic approach for DVFS dedicated to multiple clock domain processors is proposed in [29]. The approach benefits from the fact that the voltage and frequency level in each domain or functional block can be chosen independently. A multiple clock domain processor is modelled as a queue-domain network, where queue occupancies connecting two clock domains are considered as feedback signals. A proportional-integral (PI) controller is used to adapt the clock domain frequency to any workload changes.

In [6], the queue occupancy also drives PI controllers. In contradiction to the previous research, the limitation of using solely single-input queues has been lessened and multiple processing stages have been allowed. However, it is still required to have a pipelined configuration. For demonstrating the superiority of feedback techniques over the local DVFS policies for soft real-time guarantees, a realistic cycle-accurate, energy-aware, multiprocessor virtual platform is used. It is assumed that a sufficient number of deadlines is met and no further analysis or simulation of deadline misses are provided as long as the queues are not empty.

Instead of PIDs, linear quadratic regulators (LQRs) are used in [30] and demonstrated that a decreased number of feedback lines is required. However, the approach is applicable only to systems where read/write rates to queues connecting different voltage frequency islands are constant and known a priori. Therefore, it cannot deal with dynamic workloads, although some robustness to workload variations in an LQR-based approach is demonstrated in [31]. Furthermore, the controller is synthesised for a particular application and thus it cannot be efficiently applied to a system with limited a priori knowledge about future workloads. In [32], a controller based on a fractal model has been demonstrated to be superior to the PID and LQR-based ones due to the fractal nature of the considered workloads. Additionally, these works have been carried out under the assumption that the task mapping is static and the controller is used only for determining the voltage and frequency.

In [10], the problem of the hard real-time tasks allocation is investigated. For an approximate schedulability analysis, feedback mechanisms have been applied as heuristics. The tasks passing this early test have been then analysed with a time-consuming exact schedulability analysis. In contrast to this paper, the energy-saving mechanisms were not applied and the application model included hard real-time tasks only.

From the literature survey, it follows that there has been no previous work on mapping a real-time taskset dynamically to an MPSoC system while using DVFS together with control-theory based algorithms.

3. System Model

In this paper, we investigate a method to allocate (or schedule) a firm real-time taskset to an MPSoC platform. Therefore, we need a system model that covers the appropriate applications representing the workload as well as the platform architecture.

3.1. Application Model

A taskset Γ is comprised of an arbitrary number of independent tasks, $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_p\}$. Each task τ_l, $l \in \{1, \ldots, p\}$ is described with a tuple (r_l, C_l, D_l), where r_l denotes its release time, C_l is the worst-case execution time (WCET) and D_l is its absolute deadline. The timing constraints are firm. It means that infrequent deadline misses are tolerable, but the result has no value after its deadline. The taskset is not known in advance and all tasks have the same priority.

As in a typical scenario, the execution time of task τ_l is usually significantly lower than C_l [1], a non-zero slack time for this task appears, which is defined as the temporal difference between deadline D_l and the actual task completion time.

The consecutive stages of a task life-cycle are presented in Figure 1. The task τ_l is released at an arbitrary instant r_l. Then, it is dispatched to the least utilised processor at that instant. The admission control block in the selected processor performs an approximate schedulability test. If this test proves schedulability, task τ_l is admitted and then executed by a core in this processor, after setting a new P-state if required. Then, the states of both the controller and the dispatcher are updated.

The details regarding the platform architecture are provided in the following subsection.

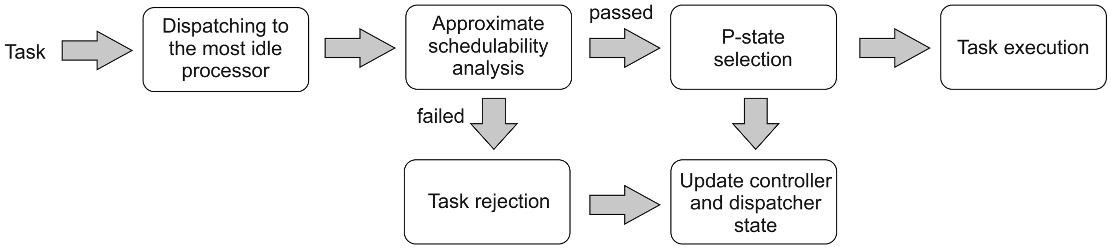
Task
Dispatching to the most idle processor
Approximate schedulability analysis
passed
P-state selection
Task execution
failed
Task rejection
Update controller and dispatcher state

Figure 1. Building blocks of the proposed approach.

3.2. Platform Model

The platform is comprised of m processors, each including n cores. Each core can be configured to work in several P-states, from P0 to P$_{max}$, which differ with respect to the computing performance and energy dissipation.

Three variants of the platforms are considered, referred to as A–C, as summarised in Table 1. In platform A, all cores in a chip operate with the same frequency and the same voltage, similarly to the Intel i7 chips. Platform B supports different frequencies in a single chip, but still the same voltage is applied across its cores, as in the contemporary AMD chips. Finally, architecture C allows both various voltage levels and frequencies in a single chip.

The general architecture of Platform A is sketched in Figure 2 (top). There is one Controller block per processor, executed on a certain core, including a single discrete-time PID controller, invoked every Δt time. The controller uses the processor's core utilisation, y, as the observed value, compares it with the setpoint, r, computes the difference between these values, e, and outputs the so-called control function u depending on the current and previous values of e, as detailed in Section 4. The Admission Control block, based on the information provided by Controller, sets an appropriate P-state of the processing Cores, one for the entire processor. In the figure, the dashed lines denote the connections used for steering P-states in the cores. Platforms B and C are outlined in Figure 2 (bottom). These platforms operate similarly to Platform A described above except for the fact that an activity of each processing Core is observed by a single Monitor, which, in turn, is connected to a Controller. Thus, the frequency (in Platform B) or both the voltage and frequency (in Platform C) can be selected independently for each processing Core.

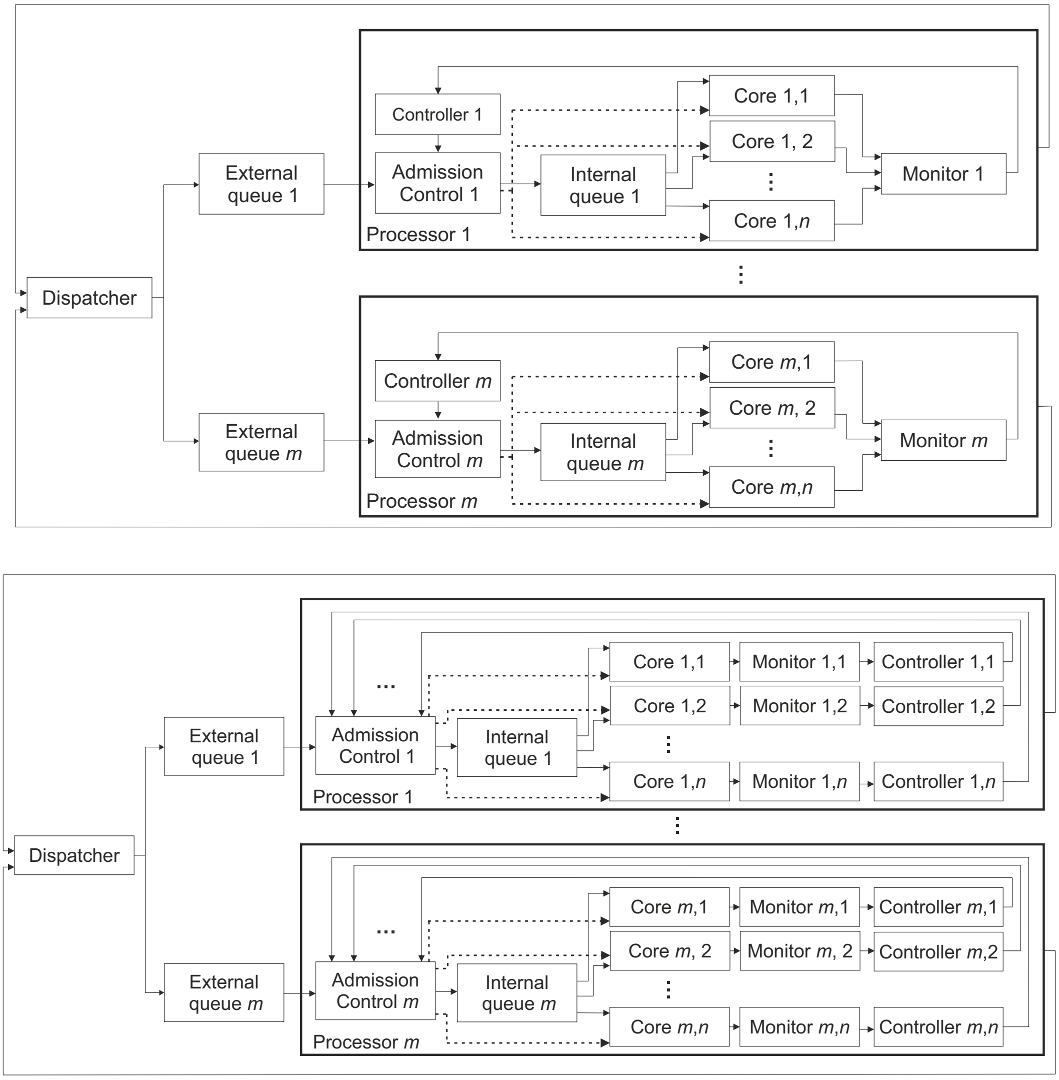

Figure 2. Distributed feedback control real-time allocation with DVFS per-chip (**top**) and per-core (**bottom**) architectures.

Table 1. Considered platform architectures.

Platform	Voltage	Frequency
A	per chip	per chip
B	per chip	per core
C	per core	per core

3.3. Problem Formulation

Given an application taskset and platform models, the problem is to execute a maximal number of firm real-time tasks before their deadlines in the highest possible (and thus the most energy-saving) P-state. As these two objectives are contradictory, some trade-off between them needs to be proposed.

4. Preliminary

A dynamic behaviour of a system can be controlled in accordance with either an open-loop or closed-loop architecture, as illustrated in Figure 3. The main difference between these approaches is that in a close-loop system, the control input ($u(t)$) partially depends on the measured output, $y(t)$, also known as controlled value of the target system, where t denotes time. Due to this feedback mechanisms, the imperfection of the Controller adjustment to the Target system can be compensated to a certain degree. As a result, the closed-loop architecture can employ less accurate models of the target system and is usually more resistant to various unexpected disturbances than its open-loop counterpart [33].

The role of the Controller is to steer the Target system using control input so that the measured output is close to the desired *reference signal* (r, also known as *setpoint*). The difference between $u(t)$ and r is referred to as *error* and is usually denoted as $e(t)$.

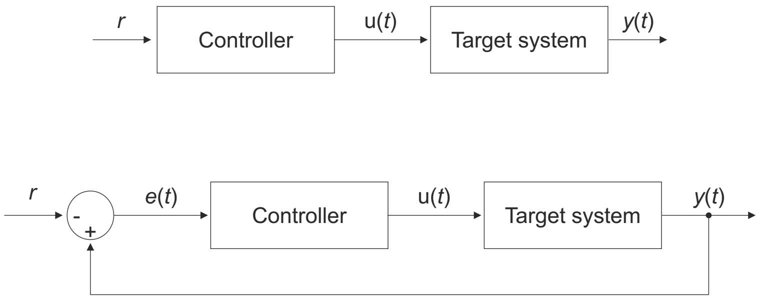

Figure 3. Control system architectures with an open-loop (**above**) and a closed-loop (**below**).

Among various controller types, the proportional-integral-derivative (PID) controller is particularly widely applied in assorted industrial control systems, including computing systems [34,35]. The control input of this controller operating in the continuous time domain is computed according to the following equation:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{d}{dt} e(t). \tag{1}$$

The controller output value, $u(t)$, is thus a sum of three components: proportional, integral and derivative. The proportional component is equal to the current error value $e(t)$ multiplied by a certain coefficient K_p. The integral component is proportional to the accumulated error over time with coefficient K_i. The derivative component is based on the current rate of error change with coefficient K_d.

In case of computer-based systems, a discretised form of a standard PID controlled is usually employed with a sampling time Δt. It can be described with equation:

$$u(t) = k_p e(t) + k_i \sum_{x=0}^{IW} e(t-x) + k_d \frac{e(t) - e(t-1)}{\Delta t}, \tag{2}$$

where the integral component is approximated with a sum of errors in so called *I-Window*, which is a time window of length $IW \cdot \Delta t$. The role of coefficients k_p, k_i and k_d is similar to their equivalents in Equation (1).

The optimal control function has to be individually determined for each control application by tuning coefficients k_p, k_i and k_d in Equation (2). There exist numerous techniques for determining approximate values of these constants. In this paper, we use the popular AMIGO (Approximate M-constrained Integral Gain Optimisation) tuning formulas as described in [33]. A detailed description of applying these formulas to a conceptually similar problem of real-time task allocation to high-performance computing clusters is presented in [36].

The derivative term in Equation (2) is highly sensitive to measurement noise, so it is often omitted in numerous practical applications [34]. The PID controller whose derivative coefficient equals 0 is referred to as PI controller. A discrete version of a PI controller is also used in the work described in this paper.

5. Proposed Approach

In case of a workload known a priori, there is a possibility of performing an exact schedulability analysis during the system design stage and then modify the platform or alter the workload itself to guarantee its timely execution even in the most adverse conditions. On the other hand, the schedulability of dynamic workloads can be analysed only during run-time. Each released task τ_l from taskset Γ can potentially jeopardise the system timeliness. It is advisable then not to admit such task that will violate its timing constraint to the processing cores and to execute other tasks before their deadlines instead. Such strategy is particularly suitable for firm real-time tasks, when there are no benefits stemming from completion a task that missed its deadline. Early rejection of such tasks not only prevent the processors from wasting their time, but also offers a possibility of early signalling of a potential deadline violation of that task, which can be used for minimising the negative impact of such situation. For example, the task can be migrated to another platform to be executed before its deadline.

In the proposed solution, the allocation of a newly released task τ_l to multi-core processors is performed by *Dispatcher*. It selects processor $Processor_j$, $j = 1, \ldots, m$ to execute the task in any of the processor's cores based on the processor *utilisation* metric, which is defined as the percentage of busy cores in $Processor_j$ with executing tasks at the current time instant. The task τ_l is then placed in the *External queue* of the processor with the lowest utilisation. When utilisations of several processors are equal to the lowest value of the utilisation, the decision is made arbitrarily among these processors. After this stage, the platform behaviour varies depending on the architecture, which is explained in the following subsections.

5.1. Platform A Scheduling

The scheduling algorithm for platform A has been described in detail in [20]. For the sake of self-consistency of this paper, it is briefly outlined below, whereas the algorithms for the two newly analysed architecture platforms, namely B and C, are detailed in the next subsection.

The path traversed by task τ_l inside a processor is indicated with solid arrows inside a processor boundary box in Figure 2 (top). The admittance control in each $Processor_j$, $j \in \{1, \ldots m\}$, is performed in accordance with the closed-loop control system architecture depicted in Figure 3. The $Controller_j$ block implements both the proportional and integral components of a discrete variant of a PID controller, described with Equation (2). The error $e_j(t)$ that is consumed by $Controller_j$ is equal to the difference between the *current utilisation* of $Processor_j$ denoted as $y_j(t)$ and the *setpoint* value of this utilisation, r, where the utilisation of $Processor_j$ is defined as the number of active cores in that processor at time point t, $t = 1, 2, \ldots$, divided by the number of cores in that processor, m. Then, the controller computes value $u_j(t)$, according to Equation (2), and provides this value to the *Admission Control* block in this processor, AC_j. Depending on the value of $u_j(t)$, AC_j decides whether a task τ_l popped from $External\ Queue_j$ is going to be executed by a core in that processor or rejected. This decision can be treated as an approximate schedulability analysis, as it is performed using the controller output value and does not involve any computational intensive exact schedulability tests, for example described in [1]. However, even if the cores' utilisation is below the setpoint, the tasks are additionally checked whether their earliest possible completion time will not violate their deadline D_l based on their worst-case execution time C_l, namely if

$$t + C_l > D_l. \tag{3}$$

If task τ_l satisfies both condition (3) and $u_j(t) \geq 0$, task τ_l is pushed in the *Internal Queue*$_j$. This queue can be very short as the tasks are rejected when the cores are busy and an idle core $Core_{j,k}$, $j = 1, \ldots, m$, $k = 1, \ldots, n$ immediately fetches a task from the internal queue. Task τ_l is executed non-preemptively up to its completion. The state of the core, i.e., its business or idleness, is observed continuously by $Monitor_j$. This block computes the utilisation of entire $Processor_j$ and sends this value to $Controller_j$. Then, the controller recomputes its output in the following time instant.

As indicated with dashed lines, the Admission Control block also steers the P-state of the entire processor. When the cores in $Processor_j$ are utilised above a certain value of the v threshold, AC_j can change P-state of this processor to increase its performance at the cost of a higher energy dissipation. Notice that, since the controller output depends not only on the current processor utilisation, but also on its previous utilisation values due to the influence of the integral component in Equation (2), a high value of controller output $u_j(t)$ indicates that the high utilisation of the processor lasts for a prolonged interval. The influence of these historic values depends both on the length of I-Window, IW, and integral coefficient k_i. In contrast, a low value of controller output $u_j(t)$, in particular below threshold $-v$, indicates a rather low utilisation of the cores in $Processor_j$ that can signalise some possibility of decreasing the processor performance to lower its energy dissipation. In the described approach, the value of the v threshold is selected after choosing the controller parameters, such as the proportional and integral constants of the controller (k_p and k_i, respectively), and the length of I-Window used in the integral component (IW). An example of determining the values of these constants is presented in Section 6.

Depending on the physical placement of the DVFS regulator in hardware, the P-state switching time can vary from the order of tens of nanoseconds in the case of on-chip regulator model to several orders of magnitude slower when off-chip regulators are employed [18] and the energy dissipation of the P-mode switching operation itself can be noticeable [19]. Consequently, applying some mechanisms that would prevent too frequent P-state switching is necessary. In the described method, the P-mode switching is prohibited when the time interval between the previous P-mode switching and the current time instant is lower than certain threshold ϕ. Hence, if the algorithm described above requires a P-state switching earlier than ϕ time after the previous alteration, this request is ignored. As the time and energy cost of P-state switching is platform-dependent, it is advisable to select this parameter experimentally, as a trade-off between the system flexibility (lower values of ϕ) and efficiency (higher values of ϕ). An influence of this parameter value on the taskset execution is discussed in Section 6.

5.2. Platforms B and C Scheduling

The proposed scheduling algorithm for platforms B and C is outlined in Algorithm 1. In principle, this algorithm is similar to its counterpart for the platform A architecture, described earlier. The primary difference is that the current P-state values are selected independently for each processing core and thus stored in an n-element vector $\mathbf{P}$ rather than a scalar. Similarly, the latest P-state switching time can be different for each processing core and thus is stored in vector $\mathbf{\Phi}$. Then, when a task is released, the processing cores in a given processor are browsed independently in a loop. In the body of this loop, variables $\mathbf{P}[k]$, $\mathbf{U}[k]$, $\mathbf{\Phi}[k]$ refer to a particular vector element and thus are indexed with the loop iterator, k. Similarly, I-Window refers to I-Window of the k-th controller in a considered processor.

The proposed algorithm for admission control is composed of two parts that are intended to be executed in parallel, as presented in Algorithm 1. The first part of the algorithm consists of lines 1–32, in which the following steps can be singled out.

Algorithm 1: Pseudo-code of the proposed admission controller functionality for per-core DVFS.

inputs : Controller output value vector $\mathbf{U}[1..n]$;
outputs : Task executing or rejection decision; New P-state of Cores;
constants : P_{max}—maximal core P-state ; υ—threshold for controller output value; ϕ—minimal time interval between consecutive P-state switchings
variables : $\mathbf{P}[1..n]$—current P-state vector; t—current time; $\mathbf{\Phi}[\mathbf{1..n}]$—time of the previous P-state change vector;

```
1  $\mathbf{P} = 0; \mathbf{\Phi} = \mathbf{0}$
2  while (true) do
3      while (External queue is not empty) do
4          Pop $\tau_l$ from the External queue;
5          for (k = 1..n) do
6              if ($\mathbf{P}[k] = 0$ and $\mathbf{U}[k] < 0$) or ($\mathbf{U}[k] < 0$ and $t \leq \mathbf{\Phi}[k] + \phi$) then
7                  if ($\mathbf{P}[k] > 0$ and $t > \mathbf{\Phi}[k] + \phi$) then
8                      $\mathbf{P}[k] = \mathbf{P}[k] - 1$;
9                      $\mathbf{\Phi}[k] = t$;
10                     Clear I-Window in the $k$-th Controller;
11                 end
12                 if ($k == n$) then
13                     Reject task $\tau_l$;
14                 end
15                 else
16                     if ($\mathbf{U}[k] < -\upsilon$ and $\mathbf{P}[k] > 0$ and $t > \mathbf{\Phi}[k] + \phi$) then
17                         $\mathbf{P}[k] = \mathbf{P}[k] - 1$;
18                         $\mathbf{\Phi}[k] = t$;
19                         Clear I-Window in the $k$-th Controller;
20                     else if ($\mathbf{U}[k] > +\upsilon$ and $\mathbf{P}[k] < P_{max}$ and $t > \mathbf{\Phi}[k] + \phi$) then
21                         $\mathbf{P}[k] = \mathbf{P}[k] + 1$;
22                         $\mathbf{\Phi}[k] = t$;
23                         Clear I-Window in the $k$-th Controller;
24                     end
25                     Push $\tau_l$ in the internal queue;
26                     Break;
27                 end
28             end
29         end
30         Wait $\Delta t$;
31     end
32 end

33 while (true) do
34     if (External queue is empty for time $\phi$) then
35         for (k=1..n) do
36             if ($\mathbf{P}[k] < P_{max}$) then
37                 $\mathbf{P}[k] = \mathbf{P}[k] + 1$;
38                 Clear I-Window in the $k$-th Controller;
39                 $\mathbf{\Phi}[k] = t$;
40             end
41         end
42     end
43     Wait $\Delta t$;
44 end
```

Step 1. Invocation and initialisation (lines 1–2, 30): The block functionality is executed in an infinite loop (line 2), activated every time interval Δt (line 30). The current P-state is set to the lowest value, P0 (i.e., the one with the highest performance—line 1), and the vector of times of the previous P-state changes, $\mathbf{\Phi}$, is set to 0 (line 1).

Step 2. Task fetching (lines 3–4): The tasks' external (FIFO) queue is checked if empty (line 3) and, if not, a task τ_l is fetched (line 4).

Step 3. Task conditional rejection (lines 6–14): The code related to this step is executed inside a loop whose iterator k ranges from 1 to n, which is the number of cores in the processor (line 5). If the output value of the k-th controller ($\mathbf{U}[k]$) is negative and the corresponding k-th core operates with the highest performance (its P-state is set to P0) or the k-th core operates below the highest performance and the interval between the previous P-state switching time (ϕ) and the current time (t) is long enough (determined by condition $t > \mathbf{\Phi}[k] + \phi$), this core is assumed to have no capacity to execute task τ_l. Consequently, if all the cores in the analysed processor have no capacity (line 12), task τ_l is rejected (line 13). Moreover, if P-state of the k-th core is different from P0 and its P-state has not been switched for at least time ϕ (line 7), P-state in the k-th core is decreased (line 8) and the k-th element of vector $\mathbf{\Phi}$, storing the previous time of the P-state alteration of the k-th core, is updated (line 9). As the previous errors in the k-th controller have been observed in a different P-state, I-Window of the k-th controller is cleared (line 10). Hence, these obsolete values do not influence future admittance decisions.

Step 4. Task conditional admittance (lines 16–27): If the output value of the k-th controller is lower than threshold $-v$, P-state of the k-th core is different from P0 and the interval between the current time (t) and the previous switching of P-state in the k-th core is long enough (line 16), the P-state is lowered (line 17) and $\mathbf{\Phi}$ is updated accordingly (line 18). In case the k-th controller output value is above threshold $+v$, P-state of the k-th core is different from the highest P-state available in the processor (P_{max}) and the previous switching of P-state in the k-th core has been done early enough (line 20), P-state of the k-th core is increased (line 21) and $\mathbf{\Phi}$ is updated appropriately (line 22). Task τ_l is sent to the Internal queue (line 25).

The second part of the algorithm is located between lines 33-44 and contains two steps only, as described below.

Step 1. Invocation (lines 33, 43): The block functionality is executed in an infinite loop (line 33), activated every time interval Δt (line 43).

Step 2. P-state conditional increase (lines 34–43): The functionality of this step is activated when no new tasks have been fetched from the External queue for interval ϕ (line 34). Under such condition, P-states of all n cores are analysed in a loop (line 35). If the core performance is not the lowest possible (P-state is different from P_{max}), the core's P-state is increased (line 37), the appropriate I-Window is cleared (line 38) and $\mathbf{\Phi}$ is updated accordingly (line 39).

The effectiveness of the proposed algorithm is experimentally evaluated in the following section.

6. Experimental Results

In this section, the efficiency of the proposed feedback-based admission control and real-time task allocation is evaluated for all three platform architectures. The experiments are performed using a cycle-accurate virtual platform developed in the SystemC language. The entities identified in Figure 2 have been implemented as custom SystemC module instances connected with the signal primitive channels, whereas both the internal and external queues are instances of the FIFO primitive channels. The SystemC module named *Core* employs ACPI parameters of a Pentium family processor. As detailed below, the *Admission Control* module implements the proposed per-core or per-chip DVFS admission controller functionality and the *Controller* module implements the discretised form of the PI controller.

Before the actual comparison between per-core and per-chip DVFS, a configuration of the parameters of the controller and admission control block has to be performed.

6.1. Controller Coefficient Setup

Controller coefficients k_p, k_i and k_d can be tuned by analysis of the corresponding open-loop (i.e., without any feedback) system output (i.e., core utilisation, $y(t)$) growth to a bursty workload. Analysis of such response is one of the most popular techniques among the control-theory-based parameter customisations [33]. The workload used as a stimuli follows the On/Off pattern in which tasks are released periodically every 5 ms during the On period lasting 500 ms, after which 500 ms-long Off period follows without any task release, as shown in Figure 4. The rationale for choosing On/Off traffic models stems from its difficulty and popularity. Some research works even claim that it is the most commonly used source model [37]. In the applied bursty workload, the computation time of each task equals 50 ms and its relative deadline is set to 75 ms. This difference between task's WCET and its relative deadline enables some level of flexibility during the task scheduling. Otherwise, if WCET equals the relative deadline, all tasks would require being started exactly at their release time to meet such tight deadline.

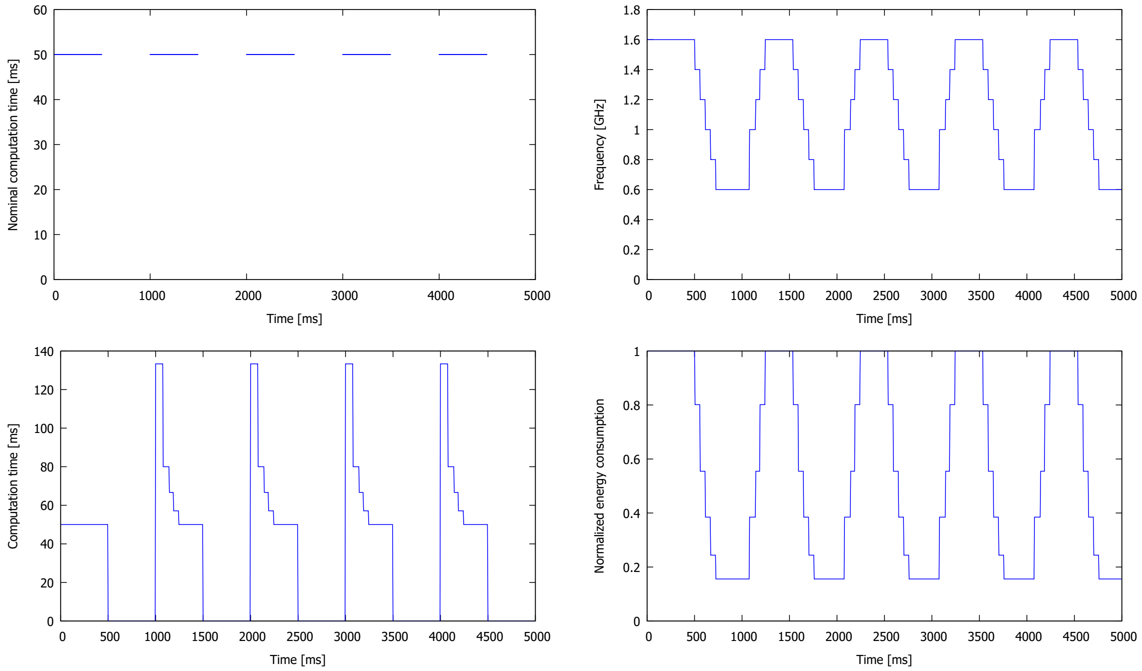

Figure 4. On/Off pattern workload (**top left**); frequency of the core executing this workload (**top right**); task computation time (**bottom left**) and normalised energy consumption (**bottom right**) in the introductory experiment.

The response of the target system confirms the accumulating (or integrating) nature of the process, i.e., the growth of the core utilisation up to the natural saturation at 100%. Due to this observed nature, the accumulating process version of Approximate M-constraint Integral Gain Optimization (AMIGO) tuning formulas can be applied to determine the PID controller coefficient values [33], similarly as shown in [35,36]. Then, the closed-loop version of a control system architecture is ready to be used.

As a baseline solution, DVFS steering capabilities have been disabled by setting threshold v to the highest possible value, denoted later as $+\infty$. Five On/Off cycles have been executed, so 500 tasks have been released in total. A system with one processor with four cores managed to execute 185 tasks before their deadlines and, thanks to the early rejection of 315 tasks by using the proposed algorithm, not a single admitted task missed its deadline.

6.2. DVFS-Related Parameter Fine-Tuning

After determining the values of the controller coefficients, a suitable value of the v threshold needs to be found. Substituting a too low value for this threshold would result in too frequent P-state switching, which is not recommended due to the additional energy and delay cost stemming from each P-state alteration. Moreover, such frequent change could prevent the proposed admission control subsystem to answer appropriately to the given workload. The reason for this is that each P-state change results in clearing the I-Window in controllers. Consequently, there would be no information available about the previous workload weight for making a decision about changing to a new P-state value. Arguably, P-state should be switched in the situation when not only the current value error $e(t)$ is high, but also similarly high values have been observed for certain time, which in turn is indicated with a high value of the integral component.

The On/Off workload pattern described previously has been applied to a platform with one processor including four cores. Several v threshold values have been evaluated, from 5 to 70, as well as the highest possible value of this parameter, denoted as ∞, practically disabling the DVFS functionality. The results of this experiment are presented in Figure 5. It is clearly visible that setting a too high value of the v threshold leads to too big of a latency in P-state switching at the beginning of the busy periods in the On/Off workload (the integral component is lower than the threshold for too long after clearing its previous values in I-Window). As a result, the number of the tasks executed before their deadlines is rather low for $30 \leq v \leq 60$. In particular, with certain v values, some tasks are admitted but executed after their deadlines, which is shown with red bars in Figure 5. In such situations, some computing resources are treated as wasted as there are no benefits from such firm real-time tasks with late completion time. It is also noticeable that, for $v \geq 70$, the delay before the allowed P-state changes is too long for the considered On/Off workload and the obtained results are the same as in the case of a disabled DVFS feature. In this situation, slightly more tasks are admitted, but much more energy is dissipated. Based on these observations, the v threshold value has been set to 10 in the experiments described below as a trade-off between the flexibility of the voltage switching and the platform performance.

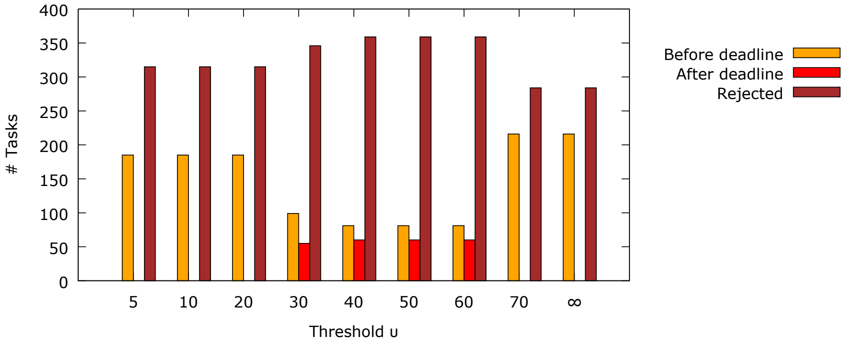

Figure 5. Total number of tasks executed before and after their deadlines and the number of tasks rejected by the admission control block with various v threshold in the introductory experiment (one processor with four cores).

The next parameter to be determined is the ϕ threshold, which specifies the minimal interval elapsing between two subsequent alterations of P-states. An experiment with the same On/Off workload has been conducted with the ϕ threshold values ranged from 25 ms to 400 ms. The highest number of tasks completed before their deadlines has been observed for three values of ϕ, namely 25 ms, 50 ms and 100 ms. For $\phi \geq 350$ ms, the obtained results have been the same as the platform without the DVFS features. In the following experiments, two values have been used: $\phi = 50$ ms and $\phi = 300$ ms.

To estimate the energy dissipation of a multi-core processor, ACPI data from one of the Pentium family processors (with Intel SpeedStep Technology) has been used. The analysed processor can operate in six P-states, from P0 (1.6 GHz, 1.484 V, 24.5 W) to P5 (600 MHz, 0.956 V, 6 W). However, the proposed solution is technology-agnostic as long as the chosen processor follows the ACPI standard. It can be, for example, applied to AMD processors as well (For example, the ACPI parameters of AMD Family 16h Series Processors can be found in AMD Family 16h Models 00h—0Fh Processor Power and Thermal Data Sheet, AMD, 2013).

In the analysed On/Off workload, the voltage and frequency levels during each On period should be high, but close to the minimum during each Off period. The results obtained in this experiment are presented in Figure 4. Notice that the frequency (and voltage, as these two values change simultaneously) does not change instantly due to the selected $\phi = 50$ ms threshold. It prevents too frequent transitions and the related additional costs of switching between P-States. At the beginning of an On period, each processor is in the least energy-consuming P-State, P5, which is characterised by the lowest clock frequency. In the analysed case, the task execution time grows from 50 ms to over 133 ms. In the next P-state, P4, this time is equal to 100 ms, whereas the tasks are executed with their nominal execution time in P0. The task execution times observed in the experiment are presented in Figure 4. Notice the difference between the execution time of the tasks in the first and the remaining On periods, caused by the cores initialisation with P0 (line 1 in Algorithm 1). If the current P-State is initialised with P5, this difference disappears. Finally, we checked a normalised energy consumption and noticed its 80% decrease during the majority of the Off periods' duration, as presented in Figure 4. Analysing energy dissipation during the entire experiment, its reduction has been equal to 43% in comparison with the corresponding system without DVFS. The latter is, however, able to process 17% more tasks before their deadlines.

As the selected parameter values lead to rather significant energy savings in this introductory experiment, it is worth analysing how a system with the same parameter values would behave while executing more sophisticated workloads or when including more processors and cores. Such experiments are presented in the remaining part of this section.

6.3. Scheduled Tasks and Energy Dissipation with per-Chip DVFS

After determining all the necessary system parameters using the On/Off workload, the efficiency of the proposed solution has been evaluted with 11 sets of 10 random workloads each. The probability distributions of the task release times and WCETs in these workloads have been based on a grid workload in an engineering design department of a large aircraft manufacturer presented in [38]. Each workload includes 100 jobs with a certain number of independent tasks, ranging from 1 to 20. All tasks belonging to a particular job with index l are released simultaneously, whereas the tasks of the subsequent job with index $l+1$ are released between $r_l + range\_min \cdot C_l$ and $r_l + range\_max \cdot C_l$, where C_l is the sum of all WCETs of tasks belonging to the l-th job released at r_l, and $range\_min, range\_max \in (0,1)$, $range\_min < range\_max$. The values of $range\_min$ and $range\_max$ are inversely proportional to the workload heaviness.

The numbers of the tasks completed before their deadlines and the numbers of the tasks rejected by the admission control block have been measured for a 2-processor platform with four cores each and a 4-processor platform with eight cores each. For both platform sizes, per-chip DVFS compliant with the platform A architecture have been compared with its counterpart without the DVFS facilities. The results are presented in Figures 6 and 7, respectively. For the 2-processor platform, 47% more tasks have been admitted with threshold $\phi = 300$ ms in comparison with threshold $\phi = 50$ ms. This result can be explained by the fact that, with $\phi = 50$ ms, P-states are switched more often. Hence, it is more likely that a core operates at a lower frequency and voltage level when a task is fetched. As P-state switching is performed gradually (lines 8, 17, 21 and 37 in Algorithm 1), the tasks are likely to be executed with a lower core performance. This reasoning is confirmed by the observed reduction of the energy dissipation. The smaller platform with $\phi = 50$ ms dissipated about 53% energy less than with $\phi = 300$ ms and 63% less in comparison with the equivalent platform with the DVFS feature disabled (i.e., with ϕ equal to $+\infty$). It can be viewed as counter-intuitive that, for lighter workloads, the number of the tasks executed before their deadlines is higher for a platform with threshold $\phi = 300$ ms than in the equivalent platform with disabled DVFS. The observed phenomenon can be explained by a deeper analysis of Algorithm 1. In a platform with disabled DVFS, each core is always in P0. The admission controller cannot then decrease the P-state when the controller output value is below 0 (which is tested in line 6) and then to clear the corresponding I-Window in the controller and, eventually, admit the task by pushing it in the internal queue (line 25). In the case of the 4-processor platform, the number of processing cores (4 × 8) is high enough to admit similar numbers of tasks regardless of the workload weight. In particular, for lighter workloads, no task is rejected nor executed after its deadline. The total numbers of tasks executed with different ϕ threshold values are almost equal (the observed differences are lower than 1.5%). However, the dissipated energy differs significantly and in the case of $\phi = 50$ ms it is almost 56% lower than with $\phi = +\infty$. The dynamic energy dissipation per task executed before its deadline, normalised to the maximal obtained value, is shown in Figure 8. In case of the 2 × 4 core platform, almost 49% and 25% less energy have been dissipated thanks to the per-chip DVFS approach with $\phi = 50$ ms and $\phi = 300$ ms, respectively. For the 4 × 8 core system, the energy dissipated in the per-chip DVFS system with both values of the ϕ parameter are similar (6% difference), but with $\phi = 50$ ms is almost 56% lower than in the case of no DVFS ($\phi = +\infty$).

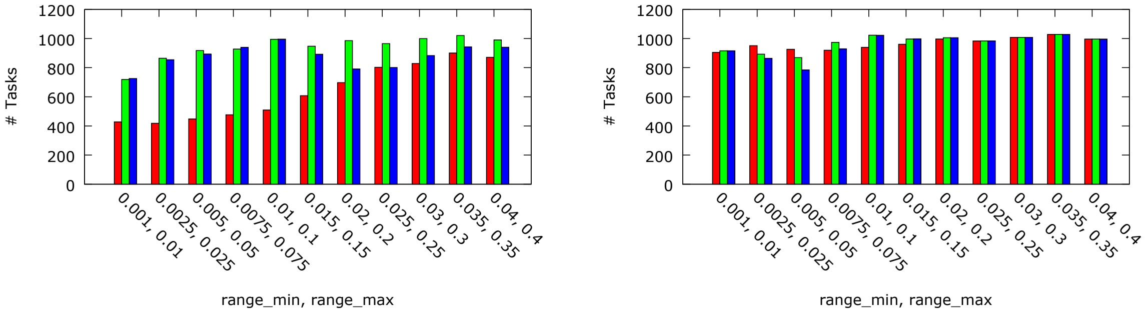

Figure 6. The number of tasks meeting their deadlines in the random workload scenarios for DVFS with $\phi = 50$ ms (red), $\phi = 300$ ms (green) and $\phi = \infty$ (blue)—two processors with four cores (**left**) and four processors with eight cores (**right**) systems.

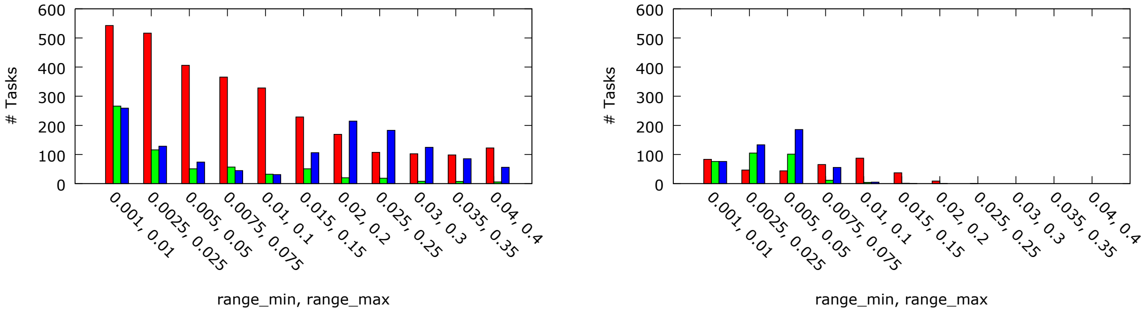

Figure 7. The number of tasks rejected by admission control in the random workload scenarios for DVFS with $\phi = 50$ ms (red), $\phi = 300$ ms (green) and $\phi = \infty$ (blue)—two processors with four cores (**left**) and four processors with eight cores (**right**) systems.

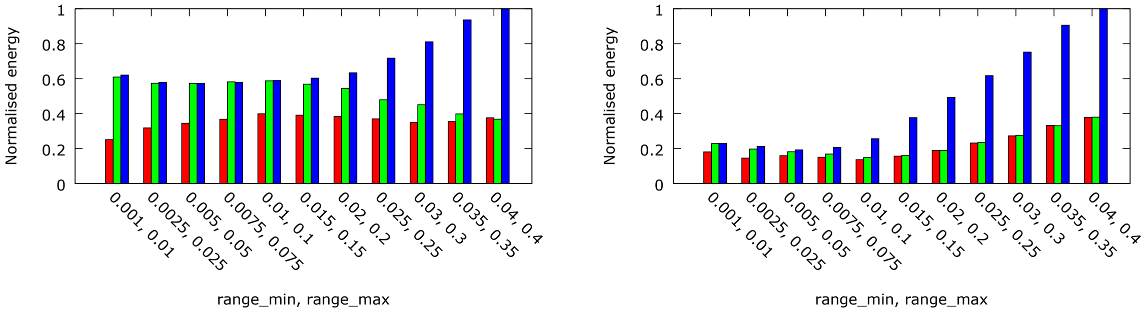

Figure 8. Normalised dynamic energy dissipation per task timely executed in the random workload scenarios for DVFS with $\phi = 50$ ms (red), $\phi = 300$ ms (green) and $\phi = \infty$ (blue)—two processors with four cores (**left**) and four processors with eight cores (**right**) systems.

6.4. Scheduled Tasks and Energy Dissipation in per-Core and per-Chip DVFS

Finally, three platform architectures A-C have been compared with respect to the number of tasks timely executed and the dissipated energy. Figure 9 depicts the number of the tasks executed before their deadlines in the random workload scenarios for platform A (left bar), platform B (middle bar) and platform C (right bar) architectures with $\phi = 50$ ms (left) and $\phi = 300$ ms (right). As both the per-core DVFS types (platform B and C) use the same scheduling algorithm outlined in Section 5.2, they execute the same number of tasks before the deadline in all cases. In case of $\phi = 50$ ms, it is noticeable that per-core DVFS (i.e., platform A) executes fewer tasks (17%) due to the higher probability that more cores are in lower P-states as the cores can be moved into slower states independently from each other. This phenomenon can be balanced by increasing the ϕ threshold and for its value $\phi = 300$ ms, the numbers of tasks executed before their deadlines are almost equal for all the considered platform architectures.

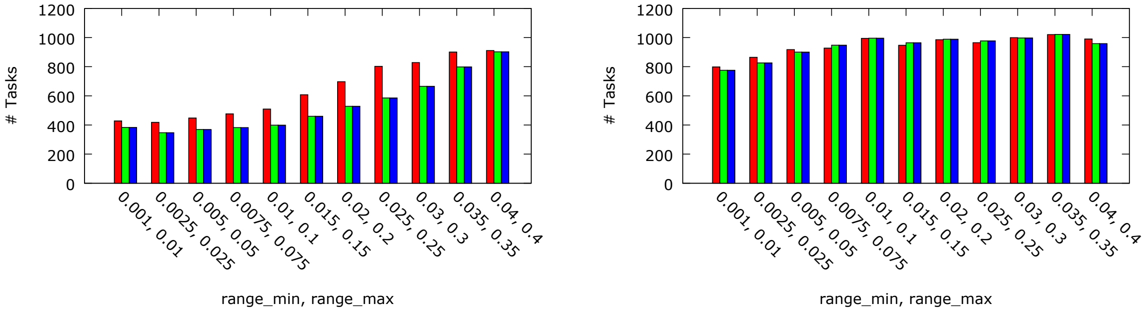

Figure 9. Tasks executed before their deadlines in the random workload scenarios for platform A (red), platform B (green) and platform C (blue) with $\phi = 50$ ms (**left**) and $\phi = 300$ ms (**right**).

In Figure 10, the normalised energy dissipated per task executed before its deadline is presented for the three platform architectures. In all cases, the difference between per-core frequency and voltage (platform C) and per-core frequency (platform B) is slight and equals about 1%. However, the difference between these platforms and platform A with both per-chip voltage and frequency DVFS is equal to 32% for $\phi = 50$ ms and almost 13% for $\phi = 300$ ms.

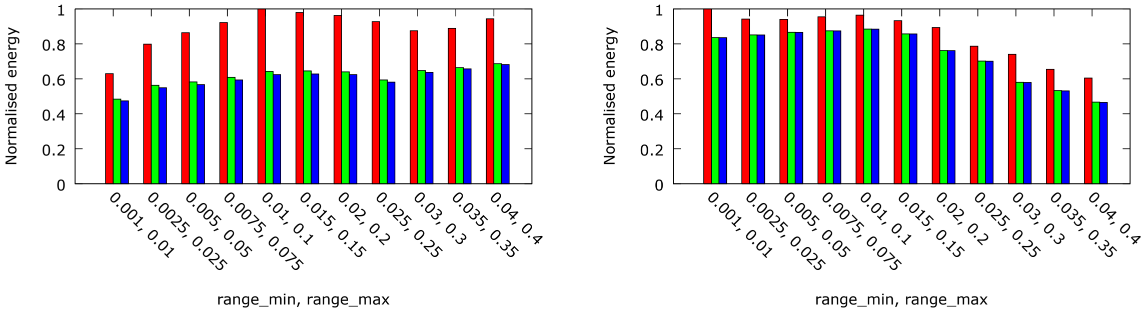

Figure 10. Normalized dynamic energy dissipation per task meeting its deadline in random workload scenarios for platform A (red), platform B (green) and platform C (blue) with $\phi = 50$ ms (**left**) and $\phi = 300$ ms (**right**).

6.5. Discussion

Due to the initial assumption of the workload not known a priori, it is difficult to design a system that simultaneously executes most real-time tasks on time and also dissipates low energy. The experimental results show clearly that the benefits of the proposed scheduling scheme depend significantly on the processor utilisation, which in turns depends on both the workload heaviness and the available computational power. Due to the presence of busy and idle periods in numerous practical situations, it is important for a system to react to urgent growth and decrease of the workload heaviness. This property has been demonstrated with the On/Off period workload in the initial experiment. The platform gradually increased the performance of its processing cores to cope with the accumulating tasks. During all On periods, the energy dissipation has grown significantly, but soon decreased after entering an Off period. In total, above 40% energy has been saved, whereas only 17% more tasks have been timely executed on the platform with no energy saving facilities. This trade-off can be even more favourable if some hints regarding the future workload are available, which would lead to better adjustment of the ϕ and v threshold values.

The experiment with random workloads has led to similar conclusions. If a workload is too heavy for a considered platform, then the influence of the threshold parameters is significant. By increasing the value of the ϕ threshold, significantly more tasks are admitted and executed before their deadlines, but the cost in terms of dissipated energy is significant. However, in case a workload is rather light for a considered platform, the majority of tasks are admitted and timely executed. In this situation, the energy savings can be particularly significant and can reach above 60% (as it has been demonstrated for the heaviest workloads for the 4x8 core system in the conducted experiments).

In all considered cases, per-chip DVFS has demonstrated higher energy dissipation (per task executed on time). This difference has been particularly visible for the workloads with medium heaviness executed on the smaller platform (2 × 4 cores), where it reached almost 40% in comparison with the corresponding per-core DVFS case. However, the difference between the platform in which both the voltage and frequency can be chosen independently per each core and the platform, in which only frequency can be set in the per-core manner, is rather negligible in all cases. Thus, the conducted experiments confirm the soundness of the MPSoC architecture proposed by AMD, where cores can operate at assorted frequencies, but all cores are supplied with the same voltage level, appropriate to the core operating at the fastest frequency.

7. Conclusions

This paper has investigated the benefits and costs of employing a feedback control scheme to firm real-time task allocation to cores in Multiprocessor Systems-on-Chips. The proposed admission control algorithm has performed an approximate schedulability analysis based on the previous platform states and rejected tasks being likely to violate their deadlines. The run-time metrics used by the admission control module were the current and prior cores' utilisations, combined by a discrete version of the proportional-integral controller. Some guidance for parameter tuning of the controller and the proposed custom governor algorithm has been provided.

Three architectures supporting dynamic voltage and frequency scaling at various MPSoC granularity levels have been used with the proposed custom governor. Even in the case of a relatively difficult On/Off workload pattern, a significant reduction of energy dissipation has been observed, but a slightly lower number of tasks have been executed before their deadlines. Similar observations have been made for random workloads based on an industrial grid. The trade-offs between the system performance and energy dissipation have been discussed for different workload weights and platform sizes. The platform architecture capable of setting the frequency per core, but the voltage per chip has been argued to be the most suitable with respect to the considered criteria.

Author Contributions: P.D. contributed to the presented task life cycle, algorithm, software implementation of the described algorithms and performing experiments. A.K.S. contributed to the plan of experiments and was a co-author of the related work section. All authors read and approved the final manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Davis, R.I.; Burns, A. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.* **2011**, *43*, doi:10.1145/1978802.1978814.
2. Casini, D.; Biondi, A.; Buttazzo, G. Semi-partitioned scheduling of dynamic real-time workload: A practical approach based on analysis-driven load balancing. In Proceedings of the 29th Euromicro Conference on Real-Time Systems (ECRTS 2017), Dubrovnik, Croatia, 28–30 June 2017; pp. 1–23.
3. Kiasari, A.E.; Jantsch, A.; Lu, Z. Mathematical formalisms for performance evaluation of networks-on-chip. *ACM Comput. Surv.* **2013**, *45*, doi:10.1145/2480741.2480755.
4. Cheveresan, R.; Ramsay, M.; Feucht, C.; Sharapov, I. Characteristics of workloads used in high performance and technical computing. In Proceedings of the 21st Annual International Conference on Supercomputing (ICS'07), Seattle, WA, USA, 17–21 June 2007; pp. 73–82.
5. Lu, C.; Stankovic, J.A.; Son, S.H.; Tao, G. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real-Time Syst.* **2002**, *23*, 85–126.
6. Carta, S.; Alimonda, A.; Pisano, A.; Acquaviva, A.; Benini, L. A control theoretic approach to energy-efficient pipelined computation in MPSoCs. *ACM Trans. Embed. Comput. Syst.* **2007**, *6*, doi:10.1145/1274858.1274865.
7. Arzen, K.E.; Robertsson, A.; Henriksson, D.; Johansson, M.; Hjalmarsson, H.; Johansson, K.H. Conclusions of the ARTIST2 roadmap on control of computing systems. *SIGBED Rev.* **2006**, *3*, 11–20.
8. Pillai, P.; Shin, K.G. Real-time dynamic voltage scaling for low-power embedded operating systems. *SIGOPS Oper. Syst. Rev.* **2001**, *35*, 89–102.
9. Ghazzawi, H.A.; Bate, I.; Indrusiak, L.S. A control theoretic approach for workflow management. In Proceedings of the 2012 17th International Conference on Engineering of Complex Computer Systems (ICECCS), Paris, France, 18–20 July 2012; pp. 280–289.
10. Dziurzanski, P.; Singh, A.K.; Indrusiak, L.S. Feedback-based admission control for hard real-time task allocation under dynamic workload on many-core systems. In Proceedings of the 29th International Conference on Architecture of Computing Systems (ARCS 2016), Nuremberg, Germany, 4–7 April 2016; pp. 157–169.
11. Tavana, M.K.; Salehi, M.; Ejlali, A. Feedback-based energy management in a standby-sparing scheme for hard real-time systems. In Proceedings of the 32nd IEEE Real-Time Systems Symposium (RTSS'11), Vienna, Austria, 29 November–2 December 2011; pp. 349–356.
12. Kidd, T. *Power Management States: P-States, C-States, and Package C-States*; Intel Software Developer Zone: Santa Clara, CA, USA: 2014.
13. Lu, C.; Stankovic, J.A.; Abdelzaher, T.F.; Tao, G.; Son, S. Performance specifications and metrics for adaptive real-time systems. In Proceedings of the 21st IEEE Conference on Real-Time Systems Symposium (RTSS'00), Orlando, FL, USA, 27–30 November 2000; pp. 13–23.
14. Kumar, P.; Nguyen, H.T.; Houghton, C.; Biermann, D.A. Providing per Core Voltage and Frequency Control. U.S. Patent 9032226, 14 April 2016.
15. Singh, A.K.; Shafique, M.; Kumar, A.; Henkel, J. Mapping on multi/many-core systems: survey of current and emerging trends. In Proceedings of the 50th Annual Design Automation Conference (DAC'13), Austin, TX, USA, 29 May–7 June 2013; 10p.
16. Lin, B.; Mallik, A.; Dinda, P.A.; Memik, G.; Dick, R.P. Power reduction through measurement and modeling of users and CPUs: Summary. In Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'07), San Diego, CA, USA, 12–26 June 2007; pp. 363–364.
17. Pallipadi, V.; Starikovskiy, A. The ondemand governor: Past, present, and future. In Proceedings of the Linux Symposium, Ottawa, ON, Canada, 19–22 July 2006; Volume 2, pp. 223–238.
18. Wonyoung, K.; Gupta, M.S.; Gu-Yeon, W.; Brooks, D. System level analysis of fast, per-core DVFS using on-chip switching regulators. In Proceedings of the 14th IEEE International Symposium on High Performance Computer Architecture (HPCA'08), Salt Lake City, UT, USA, 16–20 February 2008; pp. 123–134.
19. Gelas, J.D. *Dynamic Power Management: A Quantitative Approach*; AnandTech: New York City, NY, USA, 2010. Available online: http://www.anandtech.com/show/2919 (accessed on 20 December 2017).
20. Indrusiak, L.S.; Dziurzanski, P.; Singh, A.K. *Dynamic Resource Allocation in Embedded, High-Performance and Cloud Computing*; River Publishers: Delft, The Netherlands, 2016; 178p

21. Moreira, O.; Mol, J.D.; Bekooij, M.; van Meerbergen, J. Multiprocessor resource allocation for hard-real-time streaming with a dynamic job-mix. In Proceedings of the Real Time and Embedded Technology and Applications Symposium (RTAS'2005), San Francisco, CA, USA, 7–10 March 2005; pp. 332–341.
22. Giannopoulou, G.; Stoimenov, N.; Huang, P.; Thiele, L. Scheduling of mixed-criticality applications on resource-sharing multicore systems. In Proceedings of the International Conference on Embedded Software (EMSOFT'13), Montreal, QC, Canada, 29 September–4 October 2013; pp. 1–15.
23. Lee, Y.-H.; Daeyoung, K.; Younis, M.; Zhou, J.; McElroy, J. Resource scheduling in dependable integrated modular avionics. In Proceedings of the International Conference on Dependable Systems and Networks (DSN'2000), New York, NY, USA, 25–28 June 2000; pp. 14–23.
24. Engblom, J.; Ermedahl, A.; Sjodin, M.; Gustafsson, J.; Hansson, H. Worst-case execution-time analysis for embedded real-time systems. *Int. J. Softw. Tools Technol. Transf.* **2003**, *4*, 437–455.
25. Varma, A.; Ganesh, B.; Sen, M.; Choudhury, S.R.; Srinivasan, L.; Jacob, B. A control-theoretic approach to dynamic voltage scheduling. In Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES'03), San Jose, CA, USA, 30 October–1 November 2003; pp. 255–266.
26. Zhu, Y.; Mueller, F. Feedback EDF scheduling exploiting dynamic voltage scaling. In Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'04), Toronto, ON, Canada, 28 May 2004; pp. 84–93.
27. Soria-Lopez, A.; Mejia-Alvarez, P.; Cornejo, J. Feedback scheduling of power-aware soft real-time tasks. In Proceedings of the 6th Mexican International Conference on Computer Science, Puebla, Mexico, 26–30 September 2005; pp. 266–273.
28. Alimonda, A.; Acquaviva, A.; Carta, S.; Pisano, A. A control theoretic approach to run-time energy optimization of pipelined processing in MPSoCs. In Proceedings of the Design, Automation and Test in Europe (DATE'06), Munich, Germany, 6–10 March 2006; pp. 876–877.
29. Wu, Q.; Juang, P.; Martonosi, M.; Clark, D.W. Formal online methods for voltage/frequency control in multiple clock domain microprocessors. *SIGPLAN Not.* **2004**, *39*, 248–259.
30. Garg, S.; Marculescu, D.; Marculescu, R. Custom feedback control: Enabling truly scalable on-chip power management for MPSoCs. In Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'10), Austin, TX, USA, 18–20 August 2010; pp. 425–430.
31. Ogras, U.Y.; Marculescu, R.; Marculescu, D. Variation-adaptive feedback control for networks-on-chip with multiple clock domains. In Proceedings of the 45th ACM/IEEE Design Automation Conference (DAC'08), Anaheim, CA, USA, 8–13 June 2008; pp. 614–619.
32. Bogdan, P.; Marculescu, R.; Jain, S.; Gavila, R.T. An optimal control approach to power management for multi-voltage and frequency islands multiprocessor platforms under highly variable workloads. In Proceedings of the 6th IEEE/ACM International Symposium on Networks on Chip (NoCS'12), Copenhagen, Denmark, 9–11 May 2012; pp. 35–42.
33. Astrom, K.; Hagglund, T. *PID Controllers: Theory, Design and Tuning*; The Instrumentation, Systems and Automation Society (ISA): Research Triangle Park, NC, USA, 1995.
34. Hellerstein, J.L.; Diao, Y.; Parekh, S.; Tilbury, D.M. *Feedback Control of Computing Systems*; Wiley-IEEE Press: Hoboken, NJ, USA, 2004.
35. Janert, P.K. *Feedback Control for Computer Systems*; O'Reilly Media, Inc.: Newton, MA, USA, 2013.
36. Dziurzanski, P.; Ghazzawi, H.A.; Indrusiak, L.S. Feedback-based admission control for task allocation. In Proceedings of the 9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), Montpellier, France, 26–28 May 2014; pp. 1–5.
37. Devi, B.B. Aggregated equivalency for on-off models. In Proceedings of the IEEE Aerospace Conference, Big Sky, MT, USA, 7–14 March 2009; pp. 1–7.
38. Burkimsher, A.; Bate, I.; Indrusiak, L.S. A characterisation of the workload on an engineering design grid. In Proceedings of the High Performance Computing Symposium (HPC '14), Tampa, FL, USA, 13–16 April 2014; Society for Computer Simulation International: San Diego, CA, USA, 2014; 8p.