Feedback-Based Admission Control for Hard Real-Time Task Allocation under Dynamic Workload on Many-core Systems

Piotr Dziurzanski, Amit Kumar Singh, and Leandro Soares Indrusiak

Department of Computer Science, University of York, Deramore Lane, Heslington, York, YO10 5GH, UK.
{Piotr.Dziurzanski, Amit.Singh, Leandro.Indrusiak}@york.ac.uk

Abstract. In hard real-time systems, a computationally expensive schedulability analysis has to be performed for every task. Fulfilling this requirement is particularly tough when system workload and service capacity are not available a priori and thus the analysis has to be conducted at runtime. This paper presents an approach for applying control-theory-based admission control to predict the task schedulability so that the exact schedulability analysis is performed only to the tasks with positive prediction results. In case of a careful fine-tuning of parameters, the proposed approach can be successfully applied even to many-core embedded systems with hard real-time constraints and other time-critical systems. The provided experimental results demonstrate that, on average, only 62% of the schedulability tests have to be performed in comparison with the traditional, open-loop approach. The proposed approach is particularly beneficial for heavier workloads, where the number of executed tasks is almost unchanged in comparison with the traditional open-loop approach. By our approach, only 32% of exact schedulability tests have to be conducted. Moreover, for the analysed industrial workloads with dependent jobs, the proposed technique admitted and executed 11% more tasks while not violating any timing constraints.

1 Introduction

The vast majority of existing research into hard real-time scheduling on many-core systems assumes workloads to be known in advance, so that traditional scheduling analysis can be applied to check statically whether a particular taskset is schedulable on a given platform [5]. The hard real-time scheduling is desired in several time critical systems such as automotive and aerospace domains [8]. Under dynamic workloads, admitting and executing all hard real-time (HRT) tasks belonging to a taskset can jeopardise system timeliness so that some task deadlines may be violated. The decision of a task admittance is made by an *admission controller*. Its role is to fetch a task from the task queue and check whether it can be executed by any core before its deadline and without forcing existing tasks to miss theirs. If the answer is positive, the task is admitted, and rejected otherwise. The benefits of this early task rejection are twofold: (*i*) the resource working time is not wasted for a task that will probably violate its deadline, and (*ii*) a possibility of early signalling the lack of admittance can be employed to perform an appropriate precaution measures in order to minimize the negative impact of the task rejection (for example, to execute the task outside the considered platform).

Dynamic workloads do not necessarily follow simple periodic or sporadic task models and it is rather difficult to find a many-core system scheduling analysis that relies on more sophisticated models [5, 11]. Computationally-intensive workloads not following these basic models are more often analysed in High Performance Computing (HPC) domain, for example in [4]. The HPC community experience with these tasksets could help introducing novel workload models to many-core system schedulability analysis [5]. In HPC systems, tasks allocation and scheduling heuristics based on feedback control proved to be valuable for dynamic workloads [12], improving platform utilisation while meeting timing constraints. Despite a number of successful implementations in HPC, these heuristics are not exploited for many-core embedded platforms with hard real-time constraints.

The Roadmap on Control of Real-Time Computing Systems [1], one of the results of the EU/IST Network of Excellence ARTIST2 program, states clearly that feedback scheduling is not suitable for applications with hard real-time constraints, since feedback acts on errors. However, further research [13, 14] shows that although the number of deadline misses must not be used as an observed value (since any positive error value would violate the hard real-time constraints), observing other system's parameters, such as dynamic slack, created when tasks are executed earlier than their worst-case execution time (WCET), or core utilisation, could help in allocating and scheduling tasks in a real-time system.

Contribution: In order to address aforementioned issues, we present a novel task resource allocation process, which is comprised of the *resource allocation* and *task scheduling*. The *resource allocation* process is executed on a particular core. Its role is to send the processes to be executed to other processing cores, putting them into the task queue of a particular core. Task scheduling is carried out locally on each core and selects the actual process to run on the core. The proposed approach adopts control-theory based techniques to perform runtime admission control and load balancing to cope with dynamic workloads having hard real-time constraints. It is worth stressing that, to the best of our knowledge, no control theory based allocation and scheduling method aiming at hard real-time systems has been proposed to operate in an embedded system with dynamic workloads.

This paper is structured as follows. Section 2 describes related work. The assumptions of the considered application and platform models are enumerated in Section 3. In Section 4, the proposed runtime admission control and load balancing approach dedicated to dynamic workloads are described. Section 5 presents the experimental results to demonstrate the performance of the proposed scheme under different workload conditions. Section 6 summarizes the paper.

2 Related Work

The majority of works that apply techniques from control-theory to map tasks to processing cores offers soft real-time guarantees only, which cannot be applied to time-critical systems [12]. Relatively little work is related to hard real-time systems, where the task dispatching should ensure admission control and guaranteed resource provisions, i.e. start a task's job (a task consists of many jobs, detailed model description in Section 3) only when the system can allocate a necessary resource budget to meet its timing requirements and guarantee that no access of a job being executed to its allocated resources is denied or blocked by any other jobs [1]. Providing such kind of guarantee facilitates to fulfill the requirements of time critical systems, e.g. avionic and automotive systems, where timing constraints must be satisfied [8].

Usually hard real-time scheduling requires a priori knowledge of the worst-case execution time (WCET) of each task to guarantee the schedulability of the whole system [5]. However, according to a number of experimental results [7], the difference between WCET and observed execution time (ET) can be rather substantial. Consequently, underutilization of resources can often be observed during hard real-time system run-time. The emerging dynamic slack can be used for various purposes, including energy conservation by means of dynamic voltage and frequency scaling (DVFS) or switching off the unused cores with clock or power gating and slack reclamation protocols [13, 14].

In [6], a response time analysis (RTA) has been used to check the schedulability of real-time tasksets. This ensures meeting all hard deadlines despite assigning various execution frequencies to all real-time tasks to minimise energy consumption. In the approach proposed in this paper, RTA is also performed, but it is executed far less frequently due to the fast schedulability estimation based on controllers and thus its total execution time is shorter.

Some researchers highlight the role of a real-time manager (RTM) in scheduling hard real-time systems. In [10], it is described that after receiving a new allocation request, an RTM checks the resource availability using a simple predictor. Then the manager periodically monitors the progress of all running tasks and allocates more resources to the tasks with endangered deadlines. However, it is rather difficult to guarantee hard real-time requirements when no proper schedulability test is applied.

From the literature survey it follows that applying feedback-based controllers in hard real-time systems has been limited to determine the appropriate frequency benefiting from DVFS. According to the authors' knowledge, the feedback controller has not been yet used by an RTM to perform task allocation under dynamic workload on many-core systems.

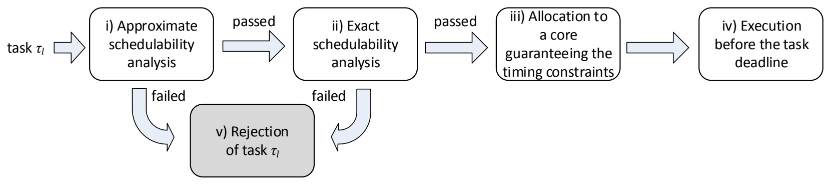

task τ_l
i) Approximate schedulability analysis
passed
ii) Exact schedulability analysis
passed
iii) Allocation to a core guaranteeing the timing constraints
iv) Execution before the task deadline
failed
failed
v) Rejection of task τ_l

Fig. 1. Building blocks of the proposed approach

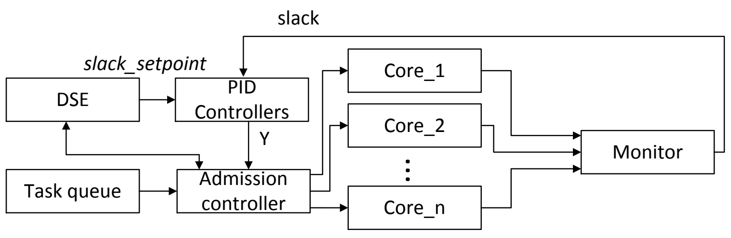


Fig. 2. Proposed many-core system architecture

3 System Model

In Figure 1, the consecutive stages of a task life cycle in the proposed system are presented. The task τ_l is released at an arbitrary instant. Then an approximate schedulability analysis is performed, which can return either fail or pass. If the approximate test is passed, the exact schedulability, characterised with a relatively high computational complexity [5], is performed. If this test is also passed, the task is assigned to the appropriate core, selected during the schedulability tests, where it is executed before its deadline.

3.1 Application Model

A taskset Γ is comprised of an arbitrary number of tasks, $\Gamma = \{\tau_1, \tau_2, \tau_3, \ldots\}$ with hard real-time constraints. The j-th job of task τ_i is denoted with $\tau_{i,j}$. If a *task* is comprised of only one *job*, these terms are used interchangeably in this paper. In case of tasks with job dependencies it is assumed that all jobs of a task are submitted at the same time, thus it is possible to identify the critical path at the instant of the task release, which can be viewed as a realistic assumption in assorted applications, e.g. industrial use cases [3]. Periodic or sporadic tasks can be modelled with an infinite series of jobs. Since the taskset is not known in advance, the tasks can be released at any instant.

3.2 Platform Model

The general architecture employed in our work is depicted in Figure 2. The system is comprised of n processing cores, whose dynamic slacks (slack vector whose length $|\text{slack}| = n$) are observed constantly by the Monitor block.

In the PID Controllers block, one discrete-time PID controller for each core is invoked every dt time. Since small sampling intervals emulate continuous time algorithms more accurately [9], the PID controllers have been decided to be activated every clock tick.

The controllers use dynamic slacks of the corresponding cores as the observed values.

The Admission controller block receives a vector of PID controllers' outputs, $\mathrm{Y} = [y_1(t), \ldots, y_n(t)]$, from the PID Controllers block. Based on its elements' values it performs, as shown in Figure 1: *(i) approximate schedulability analysis.*

If this analysis predicts that the workload is schedulable, an *(ii) exact schedulability analysis* is performed by the Design Space Exploration (DSE) block. If at the second stage the result of the task schedulability analysis is negative, the task is rejected. Otherwise it is *(iii) allocated to a core where the execution before the deadline is guaranteed* based on the schedulability analysis performed in block DSE.

3.3 Problem Formulation

Given an application and platform models, the problem is to quickly identify tasks whose hard timing constraints would be violated by the processing cores and then to reject such tasks without performing costly exact schedulability analysis. The number of rejected tasks should be reasonably close to the number of tasks rejected in a corresponding open-loop system, i.e. the system without the early rejection prediction. Meeting the deadlines for all admitted tasks shall be guaranteed.

4 Performing Runtime Admission Control and Load Balancing to Cope with Dynamic Workloads

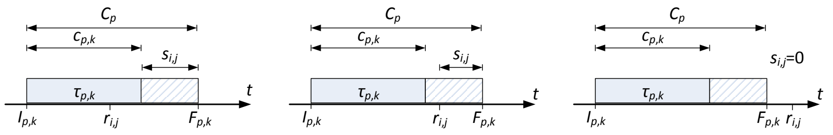


Fig. 3. Illustration of task $\tau_{i,j}$ slack in three cases from equation (1)

In dynamic workloads, admitting and executing all hard real-time (HRT) tasks belonging to a taskset can jeopardise system timeliness. The role of the admission control is to detect the potential deadline violation of a released task, τ_l, and to reject it in such a case. In doing so, the resource working time is not wasted for the task that would probably violate its deadline and early signaling of the rejection could be used for minimizing its negative impact.

The j-th job of task τ_i, $\tau_{i,j}$, is released at $r_{i,j}$, with the deadline $d_{i,j}$ and the relative deadline $D_{i,j} = d_{i,j} - r_{i,j}$. The slack for $\tau_{i,j}$ executed on core π_a, where $\tau_{p,k}$ was the immediate previous job executed by this core, is computed as follows:

$$s_{i,j} = \begin{cases} C_p - c_{p,k} & \text{if } r_{i,j} \leq I_{p,k} + c_{p,k}, \\ F_{p,k} - r_{i,j} & \text{if } I_{p,k} + c_{p,k} \leq r_{i,j} < F_{p,k}, \\ 0 & \text{if } r_{i,j} \geq F_{p,k}, \end{cases} \tag{1}$$

where $r_{i,j}$ is release time of $\tau_{i,j}$, $I_{p,k}$ - initiation time of $\tau_{p,k}$ (also known as the starting time of job execution), $c_{p,k}$ and C_p - computation time and worst-case execution time (WCET) of $\tau_{p,k}$, and $F_{p,k}$ - its worst-case completion time. A similar slack calculation approach is employed in [14]. The three possible slack cases (Equation (1)) are illustrated in Figure 3 (left, centre, right, respectively). In these figures the solid rectangle illustrates execution time (ET) of $\tau_{p,k}$, whereas the striped rectangle shows the difference between WCET and ET of this task.

The normalised value of slack of currently executed job $\tau_{i,j}$ on core π_a is computed as follows:

$$slack_a = \frac{D_{i,j} - s_{i,j}}{D_{i,j}}. \tag{2}$$

This value is returned by a monitor and compared by a PID controller with setpoint *slack_setpoint*. An error $e_a(t) = slack_a - slack\_setpoint$ is computed for core π_a, as schematically shown in Figure 2. Then the a-th output of the PID Controllers block, reflecting the past and previous dynamic slack values in core π_a, is computed with formula

$$y_a(t) = K_P e_a(t) + K_I \sum_{i=0}^{IW} e_a(t-i) + K_D \frac{e_a(t) - e_a(t-1)}{dt}, \tag{3}$$

where K_P, K_I and K_D are positive constant components of the proportional, integral and derivative terms of a PID controller, and IW is a selected length of the integral time window. Their values are usually determined using one of the well-known control theory methods, such as root locus technique, Ziegler-Nichols tuning method or many others, to obtain the desired control response and preserve the stability. In our research, we have applied Approximate M-constrained Integral Gain Optimisation (AMIGO), as it enables a reasonable compromise between load disturbance rejection and robustness [2].

The value of *slack_setpoint* is bounded between values: *min_slack_setpoint* and *max_slack_setpoint*, which should be chosen appropriately during simulation of a particular system. Similarly, the initial value of *slack_setpoint* can influence (slightly, according to our experiments) the final results. In this paper, it is initialised with the average between its minimal and maximal allowed values to converge quickly with any value from the whole spectrum of possible controller responses.

The slacks of the tasks executed by a particular processing core accumulate as long as the release times of each task are lower than the worst-time completion time of the previous task, which correspond to the first two cases in equation (1) and are illustrated in Figure 3 (left and centre). It means that the slacks of subsequent tasks executed on a given core can be used as a controller input value. However, previous values of dynamic slack are of no importance when the core becomes idle, i.e. the core finishes execution of a task and there are no more tasks in the queue to be processed, which corresponds to the third case in equation (1) illustrated in Figure 3 (right). To reflect this situation, the current value of *slack_setpoint* is provided as an error $e_a(t)$, to enhance the task assignment to this idle core (since it corresponds to the situation that the normalised slack would be twice as large as the current setpoint value, i.e. behaves in the way the task would finish its execution two times earlier than expected). Substituting this value not only positively estimates the task schedulability at the given time instant, but also influences future computation of the PID controller output, as it appears as a prior error value in the integral part in equation (3).

Algorithm 1: Pseudo-code of Admission controller involving DSE algorithm

```
inputs    : Task $\tau_l \in \Gamma$ (from Task queue)
            Vector of errors $\mathrm{Y}[1..n]$ (from PID Controller)
            Controller invocation period $dt$
            slack_setpoint decrease period $dt_1$, $dt_1 > dt$
outputs   : Core $\pi_a \in \Pi$ executing $\tau_l$ or job rejection
            Value of slack_setpoint
constants: min_slack_setpoint - minimal allowed value of slack_setpoint
            max_slack_setpoint - maximal allowed value of slack_setpoint
            slack_setpoint_add - value to be added to slack_setpoint
            slack_setpoint_sub - value to be subtracted from slack_setpoint

1  while true do
2      while task queue is not empty do
3          fetch $\tau_l$;
4          forall $\mathrm{Y}_a > 0$ do
5              if taskset $\Gamma_a \cup \tau_l$ is schedulable then
6                  assign $\tau_l$ to $\pi_a$;
7                  break;
8              end
9              if $\tau_l$ not assigned then
10                 reject $\tau_l$;
11                 if $\exists \mathrm{Y}_a : \mathrm{Y}_a > 0 \wedge$ slack_setpoint $<$ max_slack_setpoint then
12                     increase slack_setpoint by slack_setpoint_add;
13                 end
14             end
15         end
16     end
17     wait $dt$;
18 end
19 while true do
20     if slack_setpoint $>$ min_slack_setpoint then
21         decrease slack_setpoint by slack_setpoint_sub;
22     end
23     wait $dt_1$;
24 end
```

The PID Controllers block output value $\mathrm{Y} = [y_1(t), \ldots, y_n(t)]$ is provided as an input to the Admission controller block, where it is used to perform a task admittance decision. If all PID Controllers' outputs (errors) $y_a(t), a \in \{1, \ldots, n\}$ are negative, the task τ_l fetched from the Task queue is rejected. Otherwise, a further analysis is conducted by the Design Space Exploration (DSE) block to perform exact schedulability analysis. The available resources are checked according to any exact schedulability test (e.g. from [5]), which is performed for each core with task τ_l added to its taskset as long as a schedulable assignment is not found. If no resource is found that guarantees the task execution before its deadline, it is rejected.

The pseudo-code of the control strategy is presented in Algorithm 1. This algorithm is comprised of two parts, described respectively by lines 1-18 and 19-24, which are executed concurrently. The first part consists of the following steps.

– **Step 1. Invocation (lines 1, 17).**
  The block functionality is executed in an infinite loop (line 1), activated every time interval dt (line 17).
– **Step 2. Task fetching and schedulability analysis (lines 2-8).**
  All tasks present in the Task queue are fetched sequentially (line 2-3). For each task, the PID Controllers' outputs are browsed to find positive values,

which are treated as an early estimation of schedulability (line 4). If such value is found in an a-th output, an exact schedulability test checks the schedulability of the taskset Γ_a of the corresponding core π_a extended with task τ_l using any exact schedulability test (line 5), e.g. from [5]. If the analysis proves that the taskset is schedulable, τ_l is assigned to π_a (line 6). Otherwise, the next core with the corresponding positive output value is looked for.
- **Step 3. Task rejection and setpoint increase (lines 9-15).**
  If all cores have been browsed and none of them can admit τ_l due to either a negative controller output value or the exact schedulability test failure, the task τ_l is rejected (line 10). In this case, if there exists at least one positive value in the PID Controllers' output vector, the *slack_setpoint* is increased by constant *slack_setpoint_add* provided that it is lower than constant *max_-slack_setpoint* (lines 11-12) to improve the schedulability estimation in future.

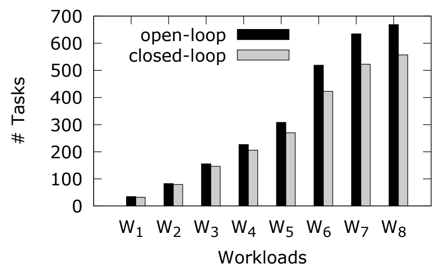
The second part consists of two steps.

- **Step 1. Invocation (lines 19, 23).**
  The block functionality is executed in an infinite loop (line 19), activated every time interval dt_1, $dt_1 > dt$ (line 23).
- **Step 2. Setpoint decrease (lines 20-21).**
  The value of *slack_setpoint* is decreased by constant *slack_setpoint_sub* (provided that it is higher than constant *min_slack_setpoint*), which encourages a higher number of tasks to be admitted in future.

5 Experimental Results

In order to check the efficiency of the proposed feedback-based admission control and real-time task allocation process, a Transaction-Level Modelling (TLM) simulation model has been developed in SystemC language. Firstly, the controller components K_P, K_I and K_D have to be tuned by analysing the corresponding open-loop system response to a bursty workload. Then two series of experiments have been performed. Firstly, workloads of various weight have been tested to observe the system behaviour under different conditions and to find the most beneficial operating region. Then industrial workloads with dependent jobs have been used to determine the applicability of the proposed approach in real-life scenarios. The details of these experiments are described below.

To tune the parameters of the controller, the task slack growth after applying a step-input in the open-loop system (i.e. without any feedback) has been analysed. This is a typical way in control-theory-based approaches [2]. As an input, a burst release of 500 tasks (each including only one single appearance job with execution time equal to 50μs) has been chosen. The modelled platform has been comprised of 3 computing cores. However, any number of tasks can be released, their execution time may vary and the number of cores can be higher, which is shown in further experiments. The obtained results have confirmed the accumulating (integrating) nature of the process, and thus the accumulating process version of AMIGO tuning formulas have been applied to choose the proper values of PID controller components [2]. Using a technique similar to [9] (chapter 15), the following constant values have been selected: *min_-slack_setpoint* $= 0.05$, *max_slack_setpoint* $= 0.95$, *slack_setpoint_add* $= 0.01$,



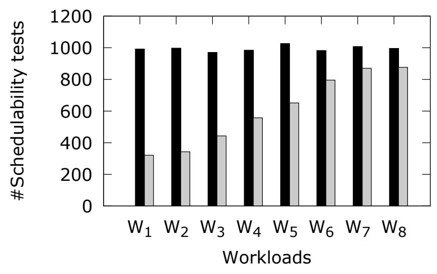


Fig. 4. Total number of tasks executed before their deadlines (left) and number of the exact schedulability test executions (right) in baseline open-loop and proposed closed-loop systems for the random workloads simulation scenario with workload sets $\mathrm{W}_1, \ldots, \mathrm{W}_8$ including 1000 tasks each

$slack\_setpoint\_sub = 0.05$, the first part of the proposed algorithm (Algorithm 1) is executed five times more often than the second one.

To check the system response to tasksets of various levels of load, eight sets, $\mathrm{W}_1, \ldots, \mathrm{W}_8$, of 10 random workloads each have been generated. Each workload is comprised of 100 tasks, including a random number (between 1 and 20) of independent jobs. The execution time of every job is selected randomly between 1 and 99μs. All jobs of a task are released at the same instant, and the release time of the subsequent task is selected randomly between $r_i + range\_min \cdot C_i$ and $r_i + range\_max \cdot C_i$, where C_i is the total worst-case execution time of the current task τ_i released at r_i, and $range\_min, range\_max \in (0, 1)$, $range\_min < range\_max$. The following parameters for pairs $(range\_min, range\_max)$ have been selected for workloads: W_1 - $(0.001, 0.01)$, W_2 - $(0.0025, 0.025)$, W_3 - $(0.005, 0.05)$, W_4 - $(0.0075, 0.075)$, W_5 - $(0.01, 0.1)$, W_6 -$(0.02, 0.2)$, W_7 - $(0.03, 0.3)$, W_8 - $(0.04, 0.4)$ to cover a wide spectrum of workload heaviness.

The numbers of executed tasks with respect to heaviness are presented in Fig. 4 (left). Both for the open-loop and closed-loop systems they are approximated better with power than linear regression (residual sum of squares is lower by one order of magnitude in case of power regression; logarithmic and exponential regression approximations were even more inaccurate). This regression model can be then used to determine the trend of executed task numbers with respect to different workload weights. Similarly, the difference between the number of admitted tasks by open and closed loop systems can be relatively accurately approximated with a power function (power regression result: $y = 960.87x^{-1.18}$, residual sum of squares $rss = 3646.06$, where x represents a workload weight computed as the total execution time of all jobs divided by the latest deadline of these jobs and y - the number of executed tasks). This relation implies that the closed-loop system admits relatively low number of tasks when the workload is light. In such lightweight condition, the number of schedulability tests to be performed is only 12% lower in the extreme case of the set W_8 (Fig. 4 (right)). Thus, there is no reasonable benefit of using controllers and schedulability estimations. In heavier loaded systems, however, the number of admitted tasks in both configurations are more balanced, and the number of schedulability test executions is significantly varied. For example, for the two heaviest considered workload sets W_1 and W_2, the schedulability tests are executed about 65% less frequently in the closed-loop system.

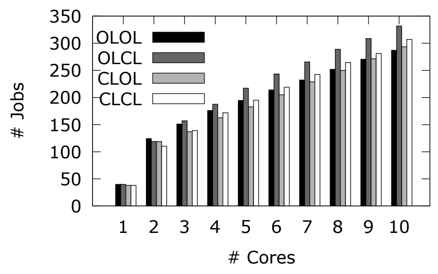
This experiment has been conducted for the number of processing cores ranging from 1 to 9. The number of executed tasks grows almost linearly with the number of cores in both configurations and the slopes of their linear regression approximations (both with correlation coefficients higher than 0.99) are almost equal. This implies that both configurations are scalable in a similar way and the difference between the number of executing tasks in open-loop and closed-loop systems is rather unvarying. The number of schedulability test executions is almost constant in the open-loop system regardless the number of cores. However, for the closed-loop configuration, it changes in a way relatively well approximated with a power regression model (power regression result: $y = 1476.29x^{-0.30}$, residual sum of squares $rss = 14216.21$, where x is the number of cores and y - the number of schedulability test executions). Since the growing number of processing cores corresponds to less computation on each of them, the conclusion is similar as in the $(range\_min, range\_max)$ variation case: the higher the load for the cores, the more beneficial is applying of the proposed scheme.

To analyse industrial workloads, 90 workloads have been generated whose dependency patterns are based on the grid workload of an engineering design department of a large aircraft manufacturer, as described in [3]. These workloads include 100 tasks of 827 to 962 jobs in total. The job execution time varies from 1ms to 99ms. Since the original workloads have no deadlines provided explicitly, relative deadline of each task has been set to its WCET increased by a certain constant (100ms).

In these workloads all jobs of any task are submitted at the same time, thus it is possible at the first stage to identify the critical path of each task and admit the task if there exists a core that is capable of executing the jobs belonging to the critical path before their deadlines. At the second stage, the remaining jobs of the task can be assigned to other cores so that the deadline of the critical path is not violated. The outputs from PID controllers can be used for choosing the core for the critical path jobs (during the first stage) or the cores for the remaining jobs (during the second stage). Four configurations can be then applied. We abbreviate them with four letter acronyms, where the two first letters denote whether the core selection for critical path tasks is done without (open loop - OL) or with (closed loop - CL) PID controllers and similarly the two remaining letters inform if the core selection for tasks outside the critical path is performed without (OL) or with (CL) PID controllers. For example, in configuration OLOL no PID controller is used and thus this configuration is treated as a baseline (only exact schedulability tests are used to select a core for a job execution).

Figure 5 (left) shows the number of jobs executed before their deadlines. The cores are scanned in a lexicographical order as long as the first one capable of executing the job satisfying its timing constraints is not found, whereas in the closed-loop configurations the tasks are checked with regards to the decreasing value of the corresponding controller outputs. Notice that the number of cores is related to the processing cores only (Core_1, ..., Core_n in Fig. 2); the remaining functional blocks (e.g. Admission controller) are realised in additional cores.

The OLOL configuration approach seems to be particularly beneficial in the systems with lower number of cores (heavier loaded with tasks). However, in the systems with more than two cores, the OLCL configuration leads to the



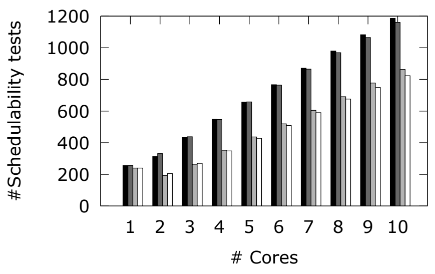


Fig. 5. Number of executed jobs (left) and number of schedulability test executions (right) for systems configured in four different ways for the industrial workloads simulation scenario

best results. Its superiority in comparison with CLCL stems from the fact that an over-pessimistic rejection of critical path jobs leads to fast rejection of the whole task. Thus the cost of a false negative estimation is rather high. Wrong estimation at the second stage usually results in choosing an idler core. The OLCL configuration admits 11% more jobs than OLOL, whereas CLCL is only slightly (about 1.5%) better than the baseline OLOL.

The main reason for introducing the control-theory based admittance is, however, decreasing the number of costly exact schedulability testing. The number of the exact test executions is presented in Figure 5 (right). Not surprisingly, the wider the usage of controller outputs, the lower is the cost of schedulability testing. The difference between OLOL and OLCL is almost unnoticeable, but the configurations with control-theory-aided selection of a core for the critical path jobs leads to significant, over 30% reduction.

From the results it follows that two configurations OLCL and CLCL dominate the others: the former in terms of number of executed jobs, the latter in terms of number of schedulability tests. Depending upon which goal is more important, one of them is advised to be selected. Interestingly, only in case of low number of processing cores, the baseline OLOL approach is slightly better than the remaining ones. For larger systems, applying PID controllers for task admissions seems to be quite beneficial.

6 Conclusions and Future Work

In this paper, we have presented a novel scheme for dynamic workload task allocation to many-cores using a control theory-based approach. Unlike the majority of similar existing approaches, we deal with workloads having hard real-time constraints that are desired in time-critical systems. Thus, we are forced to perform exact schedulability tests, whereas PID controllers are used for early estimation of schedulability. We have achieved an improved performance due to reduced number of costly scheduling test executions. For heavy workloads, up to 65% lower number of schedulability tests are to be performed when compared to typical open-loop approach, whereas the number of admitted tasks is almost equal. For industrial workloads executed on larger systems, the number of admitted tasks is higher than the open-loop approach due to the selection of idler cores for computing jobs belonging to the critical path. In future, we plan to consider heterogeneous many-core system and extend the proposed approach for mixed criticality workloads.

7 Acknowledgments

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 611411.

References

1. 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., Volume 3, Issue 3, 11–20 (2006)
2. Astrom, K., Hagglund, T.: Revisiting the Ziegler-Nichols step response method for PID control. Journal of Process Control, Volume 14, 635–650 (2004)
3. 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). Society for Computer Simulation International, Article 8 (2014)
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), pp. 73–82 (2007)
5. Davis, R.I., Burns, A.: A survey of hard real-time scheduling for multiprocessor systems. ACM Comput. Surv., Volume 43, Issue 4, Article 35 (2011)
6. Djosic, S., Jevtic, M.: Dynamic voltage scaling for real-time systems under fault tolerance constraints. In: 28th International Conference on Microelectronics (MIEL), pp. 375–378 (2012)
7. Engblom, J., Ermedahl, A., Sjodin, M., Gustafsson, J., Hansson, H.: Worst-case execution-time analysis for em-bedded real-time systems, International Journal on Software Tools for Technology Transfer. Volume 4, Issue 4, 437-455 (2003)
8. Giannopoulou, G., Stoimenov, N., Huang, P., Thiele, L.: Scheduling of mixed-criticality applications on resource-sharing multicore systems. In: International Conference on Embedded Software (EMSOFT'13), pp. 1–15 (2013)
9. Janert, P.K.: Feedback Control for Computer Systems, OReilly Media (2013)
10. Kumar, A., Mesman, B., Theelen, B., Corporaal, H., Yajun, H.: Resource Manager for Non-preemptive Heterogeneous Multiprocessor System-on-chip. In: IEEE/ACM/IFIP Workshop on Embedded Systems for Real Time Multimedia (ESTMED '06), pp. 33-38 (2006)
11. Kiasari, A.E., Jantsch, A., Lu, Z.: Mathematical formalisms for performance evaluation of networks-on-chip. ACM Comput. Surv., Volume 45, Issue 3, Article 38 (2013)
12. Lu, C., Stankovic, J.A., Son, S.H., Tao, G.: Feedback control real-time scheduling: Framework, modeling, and algorithms. Real-Time Systems, Volume 23, Issue 1/2, 85–126 (2002)
13. Tavana, M.K., Salehi, M., Ejlali, A.: Feedback-Based Energy Management in a Standby-Sparing Scheme for Hard Real-Time Systems. In: 32nd IEEE Real-Time Systems Symposium (RTSS'11), pp. 349–356 (2011)
14. Zhu, Y., Mueller, F.: Feedback EDF scheduling exploiting dynamic voltage scaling. In: 10th IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS'04), pp. 84–93 (2004)