4

Feedback-Based Allocation and Optimisation Heuristics

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 [42]. The hard real-time scheduling is desired in several time critical systems such as atomotive and aerospace domains [56]. Under dynamic workloads, admitting and executing all hard real-time (HRT) tasks belonging to a taskset can jeopardise system timeliness. The decision of task admittance is made by *admission control*. 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 with 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.

Dynamic workloads do not necessarily follow the relatively 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 [42], [67]. Computationally-intensive workloads not following these basic models are more often analysed in High Performance Computing (HPC) domain, for example in [35]. The HPC community experience with these tasksets could help introducing novel workload models to many-core system schedulability analysis [42]. In HPC systems, tasks allocation and scheduling heuristics based on feedback control proved to be valuable for dynamic workloads [82], improving platform utilisation while maintaining timing constraints. Despite a number of successful implementations in HPC community, these heuristics are to the best of our knowledge never used in many-core embedded platforms with hard real-time constraints.

The Roadmap on Control of Real-Time Computing Systems [5], one of the results of the EU/IST FP6 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 [140, 162] show 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.

The feedback-based dynamic resource allocation heuristics impose some requirements on the target system. Usually, to perform resource allocation decision one can rely on various metrics provided by the monitoring infrastructure tools and services, such as utilization and the time latency between input and output timestamps [81]. The system should also guarantee an appropriate level of responsiveness to the decisions made by the heuristics, as well as update the values of the metrics used as inputs in the algorithm. Moreover, it is important to provide the heuristic algorithm with realistic data about system workload, service capacity, worst-case execution time and average end-to-end response [84].

In order to address the 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 with 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.

4.1 System Model and Problem Formulation

In Figure 4.1 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 [42], 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.

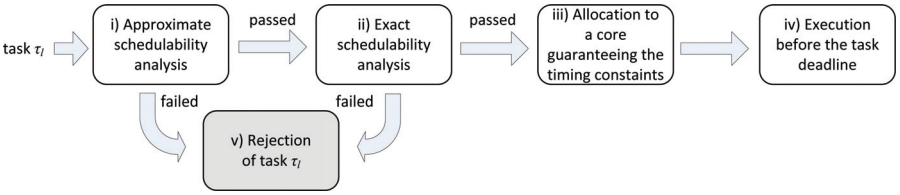

Figure 4.1 Building blocks of the proposed approach.

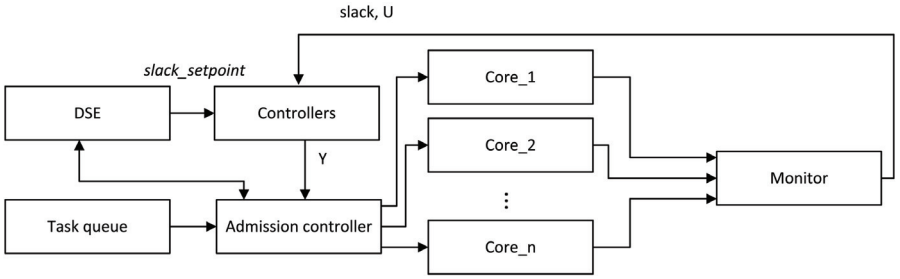

Figure 4.2 A proposed many-core system architecture.

4.1.1 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 chapter. 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. Periodic or sporadic tasks can be modelled with an infinite series of job. The taskset is not known in advance, thus the tasks can be released at any instant.

4.1.2 Platform Model

The general architecture of the proposed solution is depicted in Figure 5.3. The system is comprised of n cores, whose dynamic slacks (slack vector whose length $|\text{slack}| = n$) and busyness (vector U, $|\text{U}| = n$) are observed constantly by the Monitor block.

In the Controllers block, one discrete-time PID controller for each core is invoked every dt time. The controllers use dynamic slacks of the corresponding cores as the observed values.

The Admission controller block receives a vector of controllers' outputs, $\mathrm{Y} = [y_1(t), \ldots, y_n(t)]$, from the Controllers block. Based on its elements' values it performs, as shown in Figure 4.1, *(i) approximate schedulability analysis* of a task admittance or rejection decision. If the decision is positive, 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.

4.1.3 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 costy 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.

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

In dynamic workloads, admitting and executing all hard real-time (HRT) tasks belonging to a taskset G 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 the case. Then the resource working time is not wasted for a 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:

$$s_{i,j} = \begin{cases} C_p - c_{p,k} & \text{if } r_{i,j} \leq I_{p,k} + c_{p,k}, \\ F_{p,k} - r_{i,j} & \text{if } I_{p,k} + c_{p,k} \leq r_{i,j} < F_{p,k}, \\ 0 & \text{if } r_{i,j} \geq F_{p,k}, \end{cases} \tag{4.1}$$

where $r_{i,j}$ is release time of $\tau_{i,j}$, $I_{p,k}$ – initiation time of $\tau_{p,k}$ (also known as the job execution starting time), $c_{p,k}$ 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 [162]. The three possible slack cases (Equation (4.1)) are illustrated in Figure 4.3 (top, centre,

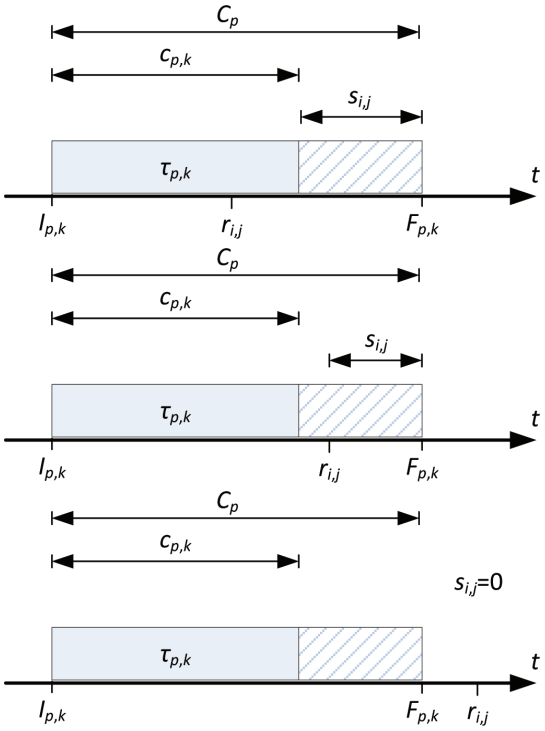

Figure 4.3 Illustration of task $\tau_{i,j}$ slack in three cases from Equation (4.1).

bottom, 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:

$$slack_a = \frac{D_{i,j} - s_{i,j}}{D_{i,j}}. \tag{4.2}$$

This value is returned by a monitor and compared by a 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 5.3. Then the a-th output of the Controllers block, reflecting the past and previous dynamic slack values in core π_a, is computed with formula

$$y_a(t) = K_P e_a(t) + K_I \sum_{i=0}^{IW} e_a(t-i) + K_D \frac{e_a(t) - e_a(t-1)}{dt}, \tag{4.3}$$

where K_P, K_I and K_D are positive constant components of the proportional, integral and derivative terms of a PID controller. 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 [8]. This method has been outlined in Chapter 3.

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 chapter, 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 (4.1) and are illustrated in Figure 4.3 (top 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 is no more tasks in the queue to be processed, which corresponds to the third case in Equation (4.1) illustrated in Figure 4.3 (bottom). 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 with 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 controller output, as it appears as a prior error value in the integral part in Equation (4.3).

The 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 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 there checked according to any exact schedulability test (e.g., from [42]), which is performed for each core with task τ_l added to its taskset as long as a schedulable assignment is not found. In our implementation, this analysis has been carried out using the interval algebra described in Chapter 2. 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 3.9. 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).

Algorithm 4.1 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 Controller)
Controller invocation period dt
$slack\_setpoint$ decrease period dt_1, $dt_1 > dt$
outputs : Core $\pi_a \in \Pi$ executing τ_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 τ_l;
4          forall Y_a > 0 do
5              if taskset Γ_a ∪ τ_l is schedulable then
6                  assign τ_l to π_a;
7                  break;
8              end
9              if τ_l not assigned then
10                 reject τ_l;
11                 if ∃Y_a : Y_a > 0 ∧ 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
```

- **Step 2. Task fetching and schedulability analysis (lines 2–8).**
  All tasks present in the Task queue are fetched sequentially (lines 2–3). For each task, the 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 [42]. 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 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.

4.3 Experimental Results

In order to check the efficiency of the proposed feedback-based admission control and real-time task allocation process, a simple 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 three series of experiments have been performed. Firstly, a heavy periodic workload has been used to observe the behaviour of the overloaded system. Due to the regularity in the workload, some convergence of the setpoint has been expected. In the second series, 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.

To tune the parameters of the controller, the task slack growth response on 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 [8]. As an input, a burst release of 500 tasks (with execution time equal to 50 μs each) has been chosen. The modelled system has been comprised of 3 computing (homogeneous) 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 [8], similarly as it has been presented in Chapter 3. With a series of trial-and-error processes, the following constant values have been selected: $min\_slack\_setpoint = 5$, $max\_slack\_setpoint = 95$, $slack\_setpoint\_add = 1$, $slack\_setpoint\_sub = 5$, the first part of the proposed algorithm (Algorithm 3.9) is executed five times more often than the second one.

During the first experiment, the system with the chosen parameters has been experimentally evaluated under a periodic workload, consisting of 900 independent jobs (i.e., each task is comprised of a single job only), one released every 5 μs, whose WCET equals to 50 μs and the relative deadline is equal to 60 μs. These parameters have been chosen appropriately to make the taskset heavy enough to saturate the system. The exact schedulability test has been performed for each task passing the early estimation based on the controller's output value. The systems with the number of processing cores ranging from 1 to 11 have been considered. The real execution time (ET) of each task varies randomly between 60% and 100% of its WCET, which results in creation of a dynamic slack. In the schedulability analysis, since the already executed tasks may influence the execution of the task whose schedulability is being tested, it is less pessimistic but still safe to provide the ET of these tasks instead of their WCET.

The regularity of the workload should cause convergence of the setpoint and decrease the variance of the normalised slack time. If the dynamic slack time normalised to task deadlines is close to 0%, it can be treated as an indication of well-chosen admission controller algorithm and the controller parameters, since it implies that the controller managed to minimize the steady-state error. The time needed to obtain this steady state indicates the responsiveness of the system, which should not be too long.

To check the system response to tasksets of various heaviness, nine sets of 10 random workloads 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 99 μs. 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 computation time of the current tasks τ_i released at r_i, and $range\_min, range\_max \in (0, 1)$, $range\_min < range\_max$. These values influence the workload heaviness which can be described with $Param$ parameter, which we define as the total execution time of all jobs divided by the latest deadline of these jobs. For example, for pair $range\_min = 0.001$, $range\_max = 0.01$, ten random workloads have been generated with $Param$ ranging from 208.65 to 237.62, with the average value $\overline{Param} = 224.52$. The average $Param$ values for the generated workloads are given in Table 4.1. The value of $\lceil Param \rceil$ can be viewed as a lower bound of the number of cores needed for computing all tasks in the workload before their deadlines. It is a rather optimistic value due to the bursty nature of the workloads and their deadlines. For example, only 71% of tasks are executed before their deadlines from a certain generated workload with $Param = 4.58$ in an open-loop 5-core system, whereas to execute all these tasks as many as 13 cores are needed.

4.3.1 Number of Executed Tasks, Rejected Tasks and Schedulability Tests

4.3.1.1 Periodic workload

The number of tasks executed before their deadlines while using both ET and WCET for schedulability analysis has been compared with the corresponding open-loop system in Figure 4.4 (top). As expected, by using actual execution time (ET), the number of tasks executed before their deadlines is slightly increased. On average, an improvement of 3.7% is achieved. The results obtained by closed-loop approaches are clearly worse than those obtained with the open-loop approach, where schedulability of each task is analysed

Table 4.1 Average $Param$ values for random workloads generated with different $range\_min$ and $range\_max$ parameters

$range\_min$	$range\_max$	$\overline{Param}$
0.001	0.01	224.52
0.0025	0.025	77.07
0.005	0.05	38.71
0.0075	0.075	25.56
0.01	0.1	18.90
0.02	0.2	9.11
0.03	0.3	6.12
0.04	0.4	4.60

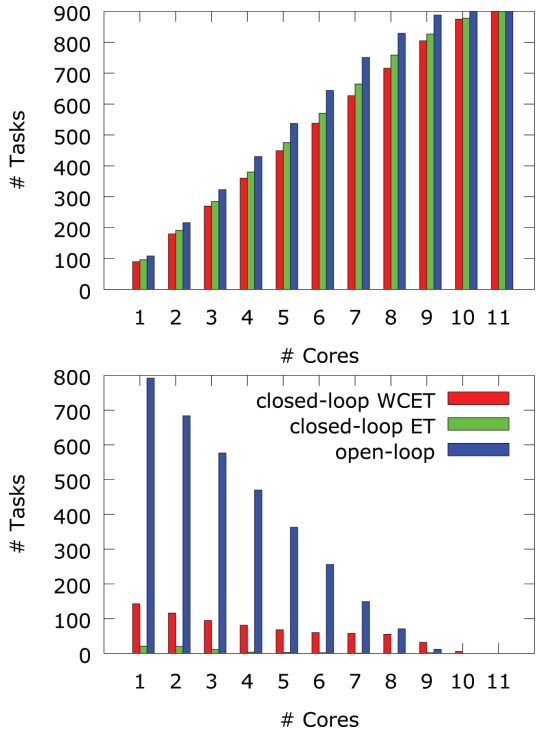

Figure 4.4 Number of executed tasks (*top*) and number of tasks rejected by the exact schedulability test (*bottom*) in closed-loop WCET, closed-loop ET and open-loop systems for the periodic task workload simulation scenario.

with an exact schedulability test only. The open-loop approach admits about 7.6% and 10.9% higher number of tasks than the closed-loop ET and WCET case, respectively. However, this improvement is achieved with a significant timing overhead. In an extreme case of one core system, 117 schedulability tests are to be conducted for the closed-loop ET case (and 233 for WCET) in comparison with 900 test executions in the open-loop system. It is important to note that only 21 exact schedulability tests for the ET case (and 143 for WCET) returned negative results, which demonstrates high accuracy of the proposed estimation scheme for open loop system. For higher number of cores, the differences are lower since each admitted task is to be checked by the exact schedulability test to ensure its hard deadlines compliance. On average, more than 38% and 34% of the schedulability tests can be omitted for the estimation based on ET and WCET, respectively. This difference is

caused by the number of tasks rejected by the exact schedulability test, which is illustrated in Figure 4.4 (bottom).

4.3.1.2 Random workload

Figures 4.5 and 4.6 present the number of tasks computed before their deadlines, rejected tasks and the number of the exact schedulability test executions with respect to the number of processing cores (ranging from 1 to 9) and average values of $Param$, respectively, for open-loop and closed-loop (ET) systems.

The numbers of executed tasks with respect to $Param$, both for the open-loop and closed-loop systems, 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 number with respect to different workload weights. Similarly, the difference between the number of admitted tasks by open and closed loop systems can be relatively accurate approximated with a power function (power regression result: $y = 960.87x^{-1.18}$, residual sum of squares $rss = 3646.06$). 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 with $\overline{Param} = 4.60$. Thus, there is no reasonable benefits 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 (i.e., with $\overline{Param}$ equal to 224.52 and 77.07) the schedulability tests are executed about 65% rarer in the closed-loop system.

The number of executed tasks grows almost linearly with the number of processing 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 good approximated with a power regression model (power regression result: $y = 1476.29x^{-0.30}$, residual sum of squares $rss = 14216.21$). Since the growing number of processing cores corresponds to less computation on each of them, the conclusion is similar as in the $Param$ variation case: the higher the load for the cores, the more beneficial is applying of the proposed scheme.

The number of tasks rejected in the open-loop systems using the exact schedulability test is considerably higher for heavier random workloads and

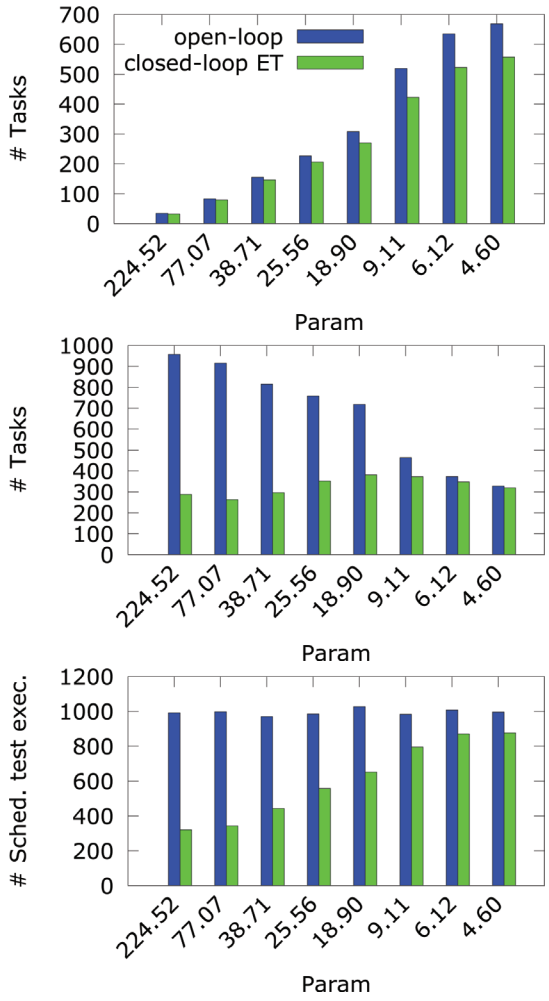

Figure 4.5 Number of tasks executed before their deadlines (*top*), the number of rejected tasks (*centre*) and number of the exact schedulability test executions (*bottom*) in baseline open-loop and proposed closed-loop ET systems for the random workloads simulation scenario with different weight of workloads.

lower number of cores, whereas for lighter random workloads or higher number of cores it is similar to the closed-loop ET system. For the closed-loop ET systems these figures illustrate the number of false positive errors of the approximate schedulability analysis, whereas for the open-loop systems it complements the number of tasks executed before deadlines.

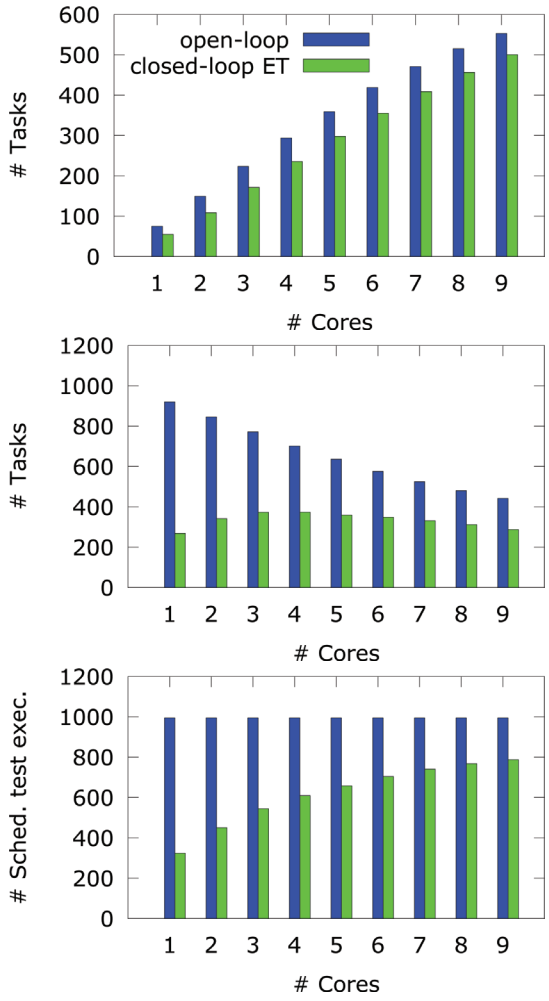

Figure 4.6 Number of tasks executed before their deadlines (*top*), the number of rejected tasks (*centre*) and number of the exact schedulability test executions (*bottom*) in baseline open-loop and proposed closed-loop ET systems with different number of processing cores for the random workloads simulation scenario.

4.3.2 Dynamic Slack, Setpoint and Controller Output

4.3.2.1 Periodic workload

Figures 4.7 shows dynamic slack, setpoint and controller outputs during the simulation for the periodic task workload executed on a system with 3 cores. In Figure 4.7 (top), the initial relative large values of the slack of three cores

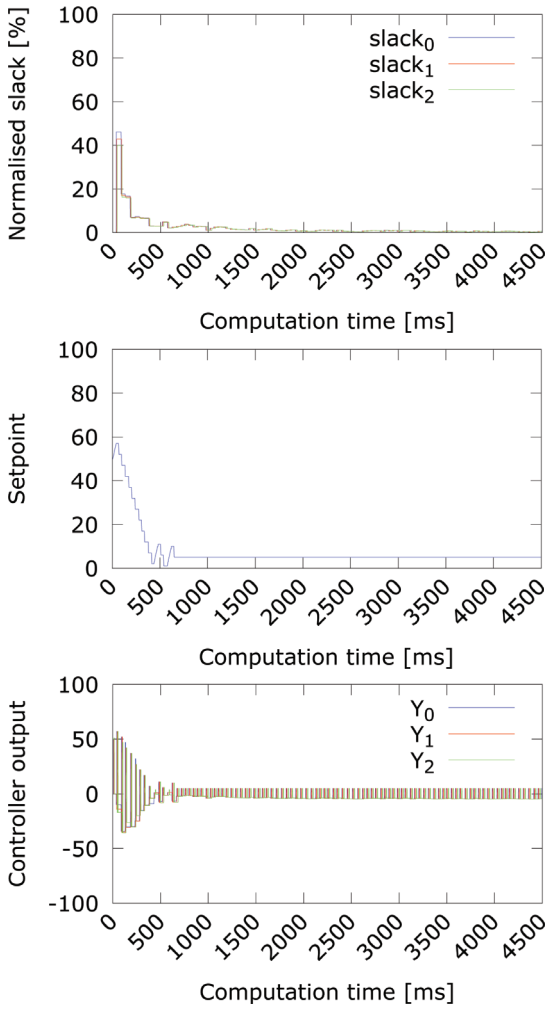

Figure 4.7 Dynamic slack (*top*), setpoint (*centre*) and controller output (*bottom*) during the simulation for the periodic task workload simulation scenario executed by a 3 core system.

(plotted with three different colors), normalised to task deadlines, equal to 46%, 42.8%, and 39.9% (the difference is caused with the random ET). The slack values decrease fast and after 530 ms none of them is higher than 5% of the task deadlines. This implies that tasks are executed relatively close to their deadlines, but never miss them. This behavior is obtained due to the exact schedulability test performed in the Design Space Exploration block and also indicates minimization of the steady-state errors by controllers. Taking into

account the initial high values of the setpoint, the time of reaching the low normalised slack time can be treated as rather short. However, in random and industrial workloads reaching any steady state is very rare due to the lack of strict periodic behaviour of the workloads, as shown later in this chapter.

The early estimation based on controller outputs does not admit too many unschedulable tasks (in this experiment only 19 such tasks have been detected by the schedulability test). It is visible in Figure 4.7 (centre), where the value of setpoint decreases from the initial value to the minimum (by the functionality of the 2nd part of the algorithm presented in Algorithm 3.9), and after 650 ms no increase is observed. It means that after this time not a single unschedulable task has been wrongly identified as schedulable by the early estimation. The initial high values of normalised slack and setpoint are also reflected in Controllers' output values (Figure 4.7 (bottom)). Every time the value of an appropriate controller output is negative, a released task cannot be executed on the corresponding processing core. Despite only a sign of the controller output is important for the task admittance, relative large values of the controller outputs denote significant variance over observed normalised slack, which may be caused with not yet stabilised value of the controller setpoint. After about 750 ms the absolute value of the controller outputs are rather low, which means that the task slacks observed in the corresponding cores are low and the workload is rather predictable as compared to the random workload (next experiment).

4.3.2.2 Light workload

In Figure 4.8, the observed run-time metrics of the closed-loop 3-core system simulation of one selected light workload (with $Param = 7.89$), taken from [23], is presented. From this particular workload, 53 tasks have been executed, 569 tasks rejected by the early estimation, and 293 tasks rejected by the exact schedulability test. In comparison with the periodic workload run-time characteristics, presented in Figure 4.7, more false positive early estimations can be observed, which is reflected in higher values in the curve depicting the setpoint value (Figure 4.8 (centre)). Since the execution time of the tasks assigned to the cores vary significantly (from 1 ms to 95 ms), the normalised slack times and consequently controller outputs differ considerably for each system core (Figure 4.8 (top, bottom)), but overall decrease trend in the normalised slack time can be noticed.

4.3.3 Core Utilization

For the periodic workload, Figure 4.9 (top) presents the total utilisation of the three cores (100% core utilisation means that all cores are busy at a particular instant). Except for the system initialisation, there is no situation that all three

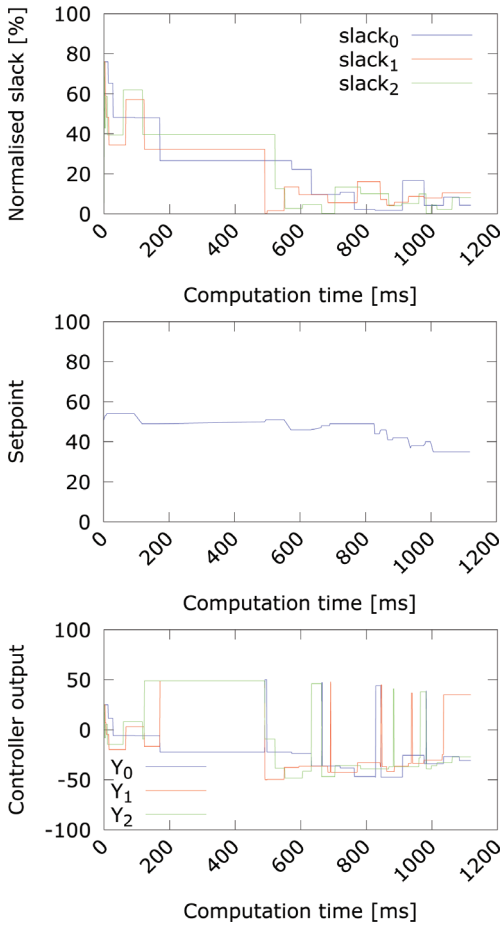

Figure 4.8 Dynamic slack (*top*), setpoint (*centre*) and controller output (*bottom*) during the simulation for the selected light workload simulation scenario executed by a 3 core system.

cores are idle. On average, the core utilisation for this simulation is equal to 83%. All three cores are balanced as the difference in their utilisations does not exceed more than 2 per mile. Similar utilisation and balance have been observed for other system configurations.

For the light workload, used also as an example in the previous subsection, a relatively long idle period of all cores can be observed (Figure 4.9 (bottom)). It is caused with the lack of task release between 10 ms and 490 ms in this particular workload. Except for this interval, it is rather difficult to observe any controller steady state, which is due to the changeable nature of the workload.

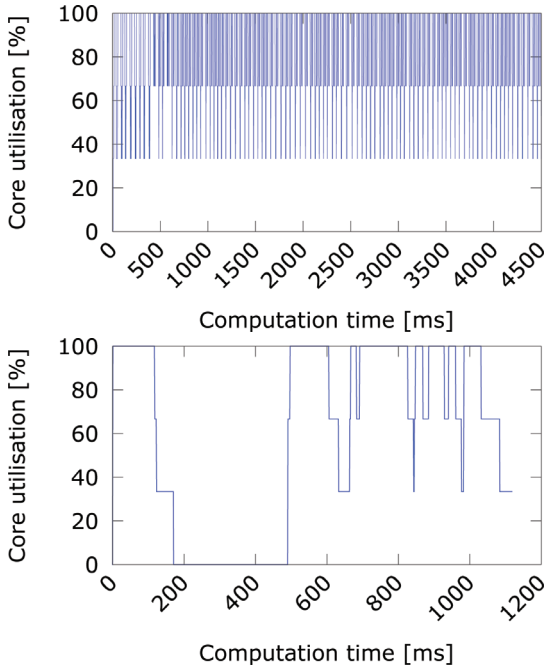

Figure 4.9 Core utilisation during the simulation for the periodic (*top*) and light (*bottom*) workload simulation scenario with 3 core system.

4.3.4 Case Study: Industrial Workload Having Dependent Jobs

To analyse industrial workloads, 90 workloads have been generated based on the grid workload of an engineering design department of a large aircraft manufacturer, as described in [23]. These workloads include 100 tasks of 827 to 962 jobs in total. The job execution time varies from 1ms to 99 ms. 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 (100 ms).

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 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, summarised in Table 4.2, can be then applied.

Table 4.2 Four configuration possibilities with respect to controllers' output usage (OL and CL stands for open-loop and closed-loop, respectively)

Core Selection for Critical Path Tasks	Core Selection for Tasks Outside the Critical Path	Configuration Abbreviation
without controllers	without controllers	OLOL (baseline)
without controllers	with controllers	OLCL
with controllers	without controllers	CLOL
with controllers	with controllers	CLCL

Figure 4.10 (*top*) shows the number of jobs executed before their deadlines. The OLOL configuration can be treated as the baseline, since no control theory elements have been applied (only exact schedulability tests are used to select a core for a job execution). The cores are scanned in a lexicographical

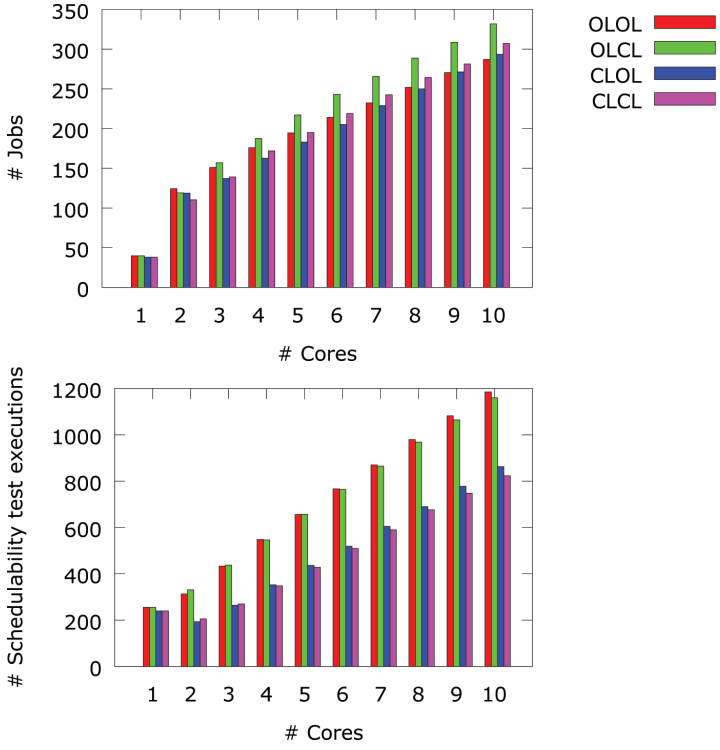

Figure 4.10 Number of executed jobs (*top*) and number of schedulability test executions (*bottom*) for systems configured in four different ways for the industrial workloads simulation scenario.

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.

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 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 4.10 (bottom). 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.

4.4 Related Work

A majority of works that apply techniques originated from control-theory to map tasks to cores offers soft real-time guarantees only, which cannot be applied to time-critical systems [82]. Relatively little work is related to the hard real-time systems, where the task dispatching should ensure admission control and guaranteed resource provisions, i.e., start a task's job, 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 [95]. 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 [56, 75].

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 [42]. However, according to a number of experimental results [48], 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 protocol [140].

In [162], the authors claim that numerous existing hard real-time schemes are not capable of adapting to dynamically changing workloads in a satisfactory manner and thus do not scale well in the average case, whereas substantial energy dissipation savings are possible. An idea of splitting each task's WCET into two intervals is presented, where the length of the first interval is equal to the predicted execution time and the remaining part is the second interval. The entire dynamic slack, accumulated from previously executed tasks, is meant to be consumed during the first interval, by executing the task with lower voltage and frequency and, consequently, lower performance. The length of this interval is determined with a proportional-integral-derivative (PID) controller. Similar approaches have been applied in [3] and [140].

In [45], 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 chapter, RTA is also performed, but it is executed far less frequent due to the fast schedulability estimation based on controllers and thus is characterised with shorter total execution time.

Some researchers highlight the role of a real-time manager (RTM) in scheduling hard real-time systems. In [74], an RTM is used together with computing resources monitoring, while schedule information are precomputed from an SDF graph statically to help guaranteeing the real-time constraints. We have extended basic ideas from their scheme to work with dynamic workloads using information gathered by the monitor. The role of RTM is also highlighted in [72]. They described that after receiving a new allocation request, it 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. In [131], an RTM exploits information about the probability of task execution time to predict the slack available for power management. It is assumed that the execution time of a task in terms of its worst-case execution time is given by a known cumulative distribution

function. The stochastic nature of this approach prevents it from application in hard real-time systems if even tiny probability (e.g., 10^{-12} [16]) of missing any deadline is not allowed.

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

4.5 Summary

In this chapter we 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 is 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 achieved an improved performance due to reduced number of costly scheduling test executions, slightly limiting the number of admitted tasks in the majority of cases. The controllers observe dynamic slack of executed tasks and aim to select the core with the lowest load.

For heavy workloads the proposed scheme achieves a better performance than using the typical open-loop approach. Up to 65% lower number of schedulability tests are to be performed, whereas the number of admitted tasks is almost equal for the heavy-loaded system and lower up to 19% with lightweight scenarios, for which the proposed scheme is less appropriate. For industrial workloads with dependent jobs executed on larger systems, the number of executed tasks using the proposed approach was even higher than the open-loop baseline system due to the selection of more idle cores for computing jobs belonging to the critical path.

Since schedulability analysis requires relatively long computation time, decreasing the number of its executions should lead to considerable computation time and energy reduction. The exact gain depends on a particular system configuration and will be evaluated in our future work. We also plan to consider heterogeneous many-core system and extend the proposed approach for mixed criticality workloads.