ATMP: An Adaptive Tolerance-based Mixed-criticality Protocol for Multi-core Systems

Abstract

The challenge of mixed-criticality scheduling is to keep tasks of higher criticality running in case of resource shortages caused by faults. Traditionally, mixed-criticality scheduling has focused on methods to handle faults where tasks overrun their optimistic worst-case execution time (WCET) estimate.

In this paper we present the Adaptive Tolerance-based Mixed-criticality Protocol (ATMP), which generalises the concept of mixed-criticality scheduling to handle also faults of other nature, like failure of cores in a multi-core system. ATMP is an adaptation method triggered by resource shortage at runtime. The first step of ATMP is to re-partition the task to the available cores and the second step is to optimise the utility at each core using the tolerance-based real-time computing model (TRTCM). The evaluation shows that the utility optimisation of ATMP can achieve a smoother degradation of service compared to just abandoning tasks.

Keywords: mixed-criticality, real-time systems, partitioned scheduling, multiprocessor systems, utility functions, fault-tolerance.

1 Introduction

Research in mixed criticality systems started with the Vestal work to provide a more flexible and effective a priori verification model for safety critical systems [28]. In fact, the increasing trend of integrating functionalities with different levels of criticality on the same platform has introduced new challenges in real-time scheduling, particularly in providing fault tolerance guarantees and in certifying such mixed criticality systems by a certification authority. However, integrating mixed criticality tasks on the same platform can be beneficial in various ways, particularly in reducing cost and energy consumption [11]. Moreover, since the beginning of this century the computer chip market has experienced what has been named as multicore revolution which is pushing all major chip producers to switch from single to multicore platforms. As a consequence, motivated by the vastly increased computational demand of real-time workloads and the trend in hardware toward multicore and multiprocessor CPUs, real-time systems are increasingly coming to be implemented upon multiprocessor platforms.

A large body of research has been performed addressing the various issues, challenges and opportunities arising from this move towards multiprocessor platforms [8]. The problems of devising algorithms to schedule a set of tasks on a multiple cores is known to be rather difficult. A well known approach is to partition the tasks into groups so that each group of tasks can be feasibly scheduled on a single processor according to some scheduling algorithm. From this point of view, the problem of partitioning a set of tasks is equivalent to the bin-packing problem [14] and it is thus reduced to determining a good partitioning scheme to map a set of tasks to a set of cores such that the number of processors used is minimum [9]. However, such heuristics are unaware of task criticalities and cannot deal with situations in which the number of cores is always insufficient.

The shift towards multi- and many-core architectures made researchers working on mixed-criticality aware of the need to design systems that cope with permanent faults. The main aim of mixed-criticality theory is that to provide higher levels of assurance to the services of highest criticality. However, the focus of the majority of research is limited to transient faults, in particular to transient faults caused by different levels of *Worst-Case Execution Time* (WCET) assurances and overruns. So far, the challenges of implementing mixed-criticality systems to tolerate permanent faults, i.e., faults whose presence is assumed to be continuous in time, has largely remained out of focus. This is a key issue to face since permanent faults such as a permanent processor failures can render the system

even useless or leading it to unsafe states. One of traditional solutions to processor failure is to use resource redundancy such as physical hardware replication and multiple software versions. Recently, Thekkilakattil et al. pointed out the need to design mixed criticality systems that can cope with permanent faults [25]. He presented a taxonomy of spatial redundancy techniques to tolerate permanent faults and identified how mixed-criticality architectures can be implemented when using spatial redundancy.

However, such solutions are usually popular in general purpose multicomputer or distributed systems but the embedded systems stringent design constraints often preclude the usage of such expensive resource redundancy. In fact, the adoption of redundancy in embedded system design is often not economical due to the *Size, Weight and Power* (SWaP) requirements. An alternative software solution to hardware resource replication, especially if we consider the always increasing trend in moving from uniprocessor to multi and many core architectures, is to migrate tasks assigned to a faulty processor to others still available when a failure is detected [18]. In such approaches, the timely detection of processor failures is crucial since the migrated tasks should be restarted on newly allocated processing elements as soon as possible so to minimise performance degradation.

Therefore, there is the need of heuristics to partition mixed-criticality tasks to deal with such permanent shortages of computing resources. An important issue arising is to decide what tasks to drop if the reduced available computing resources cannot cope anymore with the workload represented by the initial set of tasks. We propose an algorithm that, in case of sudden failure of some cores, re-allocate the initial task set on the remaining cores and optimises the frequency rate of the new resulting partitioned set of tasks to make them schedulable. The heuristics drops first tasks with least criticality and utility in which it is not possible to perform an adequate period adjustment. The approach is adaptive since, in case some processing elements become available again at runtime, a further mapping of tasks to cores and optimisation is made.

This article contains the following contribution:

1. Introducing a criticality and utility-aware adaptation method (ATMP) that maximises the utility on each core by adjusting the periods of tasks within their tolerance range.

2. Evaluating this adaptation method by comparing it with the standard utility-agnostic approach that simply drops tasks according to their utilisation factor to adjust the system load.

2 Related Work

Since the mixed-criticality task model was first proposed by Vestal in 2007 [29], the model has been amply developed and extended. Burns and Davis published a review on mixed criticality systems [4] containing an historical introduction of the topic, the challenges to face to better develop the mixed-criticality systems and the future directions to investigate.

The current trend towards the integration of cores into multi-core architectures allows to tasks having different criticalities to run on the same platform. This raises new challanges due both to the potential task interference among mixed-criticality tasks and to the verification and certification of platform subsystems. From this point of view, a holistic architecture for the seamless mixed-criticality integration encompassing distributed systems, multi-core chips, operating systems and hypervisors is still an open research problem. Obermaisser et al. describe the state-of-the-art of mixed-criticality systems and discuss the ongoing research within the European project DREAMS on a hierarchical mixed-criticality platform with support for strict segregation of subsystems, heterogeneity and adaptability [22].

The *Integrated Dependable Architecture for Many Cores* (IDAMC) platform was introduced to run multiple mixed-critical applications on a single multi-core platform [27, 21]. IDAMC is a NoC tiled architecture that provides spatial and temporal isolation. It supports safe sharing of resources, a transparent mapping of applications to available resources, isolation of highly critical tasks against faulty low critical tasks on a shared platform.

Su et al. analyse the performances of the *Elastic Mixed-Criticality* (E-MC) approach on systems with multiple identical cores that can share different levels of on/off-chip caches [24]. E-MC was first introduced together with the *Early-Release EDF* (ER-EDF) on uniprocessor systems to improve the service level provided for low-criticality tasks. The authors first investigate the schedulability of E-MC tasks under various well-known task-to-core mapping heuristics and then compare the results with the Global EDF-VD scheduler. They show that the E-MC with ER-EDF on multi-core systems improves the service levels of low-criticality tasks while Global EDF-VD may negatively affect them by canceling most of their task instances at runtime, especially for systems with more cores.

Legout et al. propose the LPDPM-MC approach [19] to reduce the energy consumption in multiprocessor mixed-criticality embedded systems by continuining to guaranteeing that high-criticality tasks meet their

deadlines. Since tasks usually do not use all their WCET estimates and low-criticality tasks are assured at a lower level, such approach uses part of the time budjet of low-criticality tasks to find an appropriate trade-off between the number of missed deadlines of low-criticality tasks and energy consumption. The approach uses the LPDPM algorithm [20] to minimize the static energy consumption via linear programming.

Thekkilakattil et al. propose a fault-tolerant approach to mixed-criticality real-time scheduling that considers the recommendations given by the hardware reliability studies like *Functional Hazard Analysis* (FHA) and *Zonal Hazard Analysis* (ZHA) to improve the overall system reliability and safety [26]. FHA and ZHA are usually used for safety critical systems to ensure that the proposed redundancies on the hardware components, e.g., wires and communication sub-systems, indeed exist. Such approach for scheduling mixed criticality real-time systems aims to provide real-time guarantees for the critical tasks offline and to ensure flexibility for the non-critical tasks.

Burns et al. adapted the traditional cyclic executive scheduling on multi-core systems to handle tasks having up to five criticalities [5]. The authors consider both partitioned and global scheduling schemes and criticality monotonic as priority assignment. Because of this, at any instant, all the processors are only allowed to execute code of the same criticality level as this rules out the possibility that less critical code interferes with the execution of more critical code in accessing shared resources [10]. The authors have studied their approach in partitioning scheduling, by using common heuristics to map application tasks to the multi-core cyclic executives, and in global scheduling, by proposing a polynomial-time sufficient schedulability test to determine whether a given mixed-criticality system is schedulable together with an algorithm to build a feasible schedule. Lastly, they also estimate the reduction in schedulability that arises from the requirement that only code of the same criticality executes at the same time on different cores.

Izosimov and Levholted have presented a new metric to design and assess mixed-criticality multi-core systems without changing the development flow and practice [13]. The primary goal in development of such metric was to provide a tool for engineers and safety managers in taking decisions with respect to the mixed-criticality and help to justify and judge a particular solution for safety-critical system design. The proposed mixed-criticality metric balances the reduction in severity of faults against implications on reduction in performance and increase in system complexity.

Kirner et al. used utility functions to optimise performances and to allow for reconfiguration at runtime in case of permanent failures in mixed-criticality systems [15, 16]. Kirner proposed a model called *Tolerance-based Real-Time Computing Model* (TRTCM), that exploits the range between the latency where the service utility becomes zero and the latency chosen as technical deadline to smoothly degrade the quality of high critical services in case of resource shortage till a level that is still acceptable. This feature is not supported by existing mixed-criticality approaches that focus on services guarantees at different certification level, rather than on system utility in presence of faults. TRTCM allows to consider different performance parameters via utility functions such as latency, throughput and jitter and to optimise the *Quality of Service* (QoS) by maximising the overall system utility in case of standard and mixed-criticality real-time services. Such approach has been further developed for adaptation of mixed-criticality systems with periodic task sets on uniform multiprocessors [17].

3 System Model and Assumptions

In the following section we describe the tolerance-based mixed-criticality system model used in this paper. We assume a mixed-criticality system, which consists of multiple tasks that could have different levels of criticality. Each task τ_i of a task set τ is defined as follows:

$$\tau_i = \langle p_i, d_i, \vec{c_i}, l_i, u_i \rangle \tag{1}$$

p_i represents the period of task τ_i.

d_i is the relative deadline of task τ_i. We assume implicit deadlines, i.e. $d_i = p_i$. (Note that such assumption is only chosen for a concrete scheduling test in our implementation, but it is not a requirement of our optimisation method.)

l_i is the criticality level of task τ_i with $l_i > 0$. A higher value of l_i means a higher level of criticality. The vector $\vec{l}$ is used to represent all possible criticality levels in a system: $\vec{l} = (l_1, \ldots, l_k)$, with l_1 being the minimum and l_k being the maximum possible criticality level.

u_i is the relative utility of task τ_i with $0 \leq \tau_i \leq 1$. The calculation of u_i is described in Section 3.1. We also use an *absolute utility* U_i, which is calculated as $U_i = u_i \cdot l_i$.

$\vec{c}$ is a vector of WCET estimates, with one WCET estimate per criticality level l_i and the additional

constraint $l_x < l_y \implies c_i(l_x) \leq c_i(l_y)$. Note that $c_i(l_x)$ is only defined for $l_x \leq l_i$.

The individual instances of a task at runtime are called jobs. A job j is described by the following tuple:

$$j = \langle a, p, d, et, \vec{c}, l \rangle$$

where a_i is the arrival time and et is the actual execution time. The entries $p, d, \vec{c}$ and l are inherited from the job's task structure.

3.1 Utility Function

In our tolerance-based mixed-criticality model the period p_i of a task τ_i is not a given constant, but can be chosen by our optimisation method within a certain interval that is specific for each individual task. However, the chosen value p_i determines the utility u_i. To be able to calculate the utility of a task, we assume that each task $\tau_i \in \tau$ has some additional utility parameters up_i:

$$up_i = \langle p_{prim,i}, p_{tol,i}, u_{tol,i} \rangle \quad (2)$$

$p_{prim,i}$ is the *primary period* of task τ_i, representing the optimal execution rate. For any period $p \leq p_{prim,i}$ the relative utility is one: $u_i = 1$.

$p_{tol,i}$ is the *tolerance period* of task τ_i, which is the maximum period still tolerable for task τ_i.

$u_{tol,i}$ is the *tolerance utility* of task τ_i, which is the relative utility at period $p_{tol,i}$.

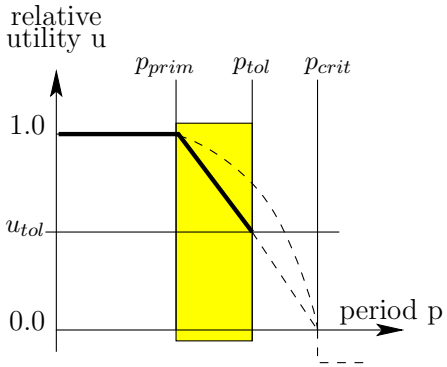


Figure 1: Utility function to calculate relative utility based on chosen period

Figure 1 shows how the utility parameters up_i describe the utility function of a task τ_i. The modelled tolerance section of the utility function is of linear shape and it is used to smoothly adjust, i.e. degrade or speed up according to the circumstances, the task arrival rates at runtime. Figure 1 also contains the *critical period* (indicated as p_{crit}) that represents the arrival rate for which the task utility becomes zero. More details on the tolerance-based real-time model can be found in [17].

The key concept of the tolerance range is that it allows to tune the task period within tolerable utility values. Based on this, the possible load of a task τ_i is within its so-called primary load $load_{prim,i}$ and its tolerance load $load_{tol,i}$:

$$load_{prim,i} = \frac{c_i}{p_{prim,i}}, \quad load_{tol,i} = \frac{c_i}{p_{tol,i}} \quad (3)$$

where c_i represent the non-conservative WCET estimates of an individual task. Consequently, this allows to also vary the total system load within $load_{prim}$ and $load_{tol}$:

$$load_{prim} = \sum_{\tau_i \in \tau} \frac{c_i}{p_{prim,i}}, \quad load_{tol} = \sum_{\tau_i \in \tau} \frac{c_i}{p_{tol,i}} \quad (4)$$

4 Optimisation Method

The *Adaptive Tolerance-based Mixed-criticality Protocol* (ATMP) consists of the following two main parts:

1. Tasks are first sorted according to decreasing criticality. Then, the partitioning of tasks to cores is made as in Algorithm 1, i.e., highest criticality tasks are selected and assigned to the core with least load allocated.

2. If a task set allocated to a specific core is schedulable, then it is processed by the underlying scheduler otherwise a binary search heuristics with linear programming optimisation is performed on each core as showed in Algorithm 2.

The main feature of ATMP is that to exploit the tolerance range described in Section 3 to optimise the tasks' periods. Each task has its own tolerance range $[p_{prim}, \ldots, p_{tol}]$ at which corresponds a utility range $[1, \ldots, u_{tol}]$ and its runtime adaptation capability is classified according to the relationship between its tolerance range and its tolerance utility as in Figure 2. The higher is the tolerance utility u_{tol} corresponding to p_{tol} and the larger is the tolerance range extent, the better the runtime adaptation will be. Therefore, the adaptation at runtime is made considering first tasks that have a higher utility corresponding to the p_{tol} value and a larger tolerance range extent.

According to the specific needs, the systen designer can set a tolerance utility value as well as a tolerance

range extent value under which the period optimisation could be considered not useful anymore. Then, among tasks with least criticality, ATMP drops tasks according to their capability adaptation, i.e. first are dropped tasks corresponding to Figure 2.d), then tasks in Figure 2.c), next tasks in Figure 2.b) and lastly tasks in Figure 2.a).

If a task set allocated to a specific core is deemed to be not schedulable, then a binary search is perfomed for a predefined number of times by modifying a copy of the partitioned task set assigned to such core. Every time, the binary search finds an lm value to use as load upper bound for the set of tasks allocated. The algorithm checks wheter such task set has a tolerance load greater then the lm value found by binary search. If the task set has tolerance load greater than lm, then the algorithm drops the first least criticality task with worst adaptation capability (Figure 2). This goes on till the task set tolerance load becomes not grater than lm.

Once a suitable task set with tolerance load not greater than lm has been found, then the ATMP protocol exploits the individual tasks tolerance ranges to find the best arrival rate for each task such that the utility is maximised. If such task set consists of just one task, then it is assumed to have a load less than or equal 100% and thus it is schedulable by default according to its primary period. Binary search continues on the upper half to find a task set with better tolerance load. On the other hand, if the resulting task set consists of more than one task, then it is used to create an LP problem according to TRTCM model published in [17] with lm used as upper bound for the resource constraint. The LP optimisation gives the optimised periods for each task. A schedulability test is perfomed to check wheter such optimised task set is feasible. Our method is independent from the specific feasibility test used. However, we use the *AMCrtb* schedulability analysis since we have considered task sets with two criticality levels [3]. Whenever a schedulable optimised task set with higher tolerance load is found, it is stored and binary search continues on the upper half to find a better optimised task set. If the optimised task set is not schedulable, then binary search continues in the lower half.

5 Experimental evaluation

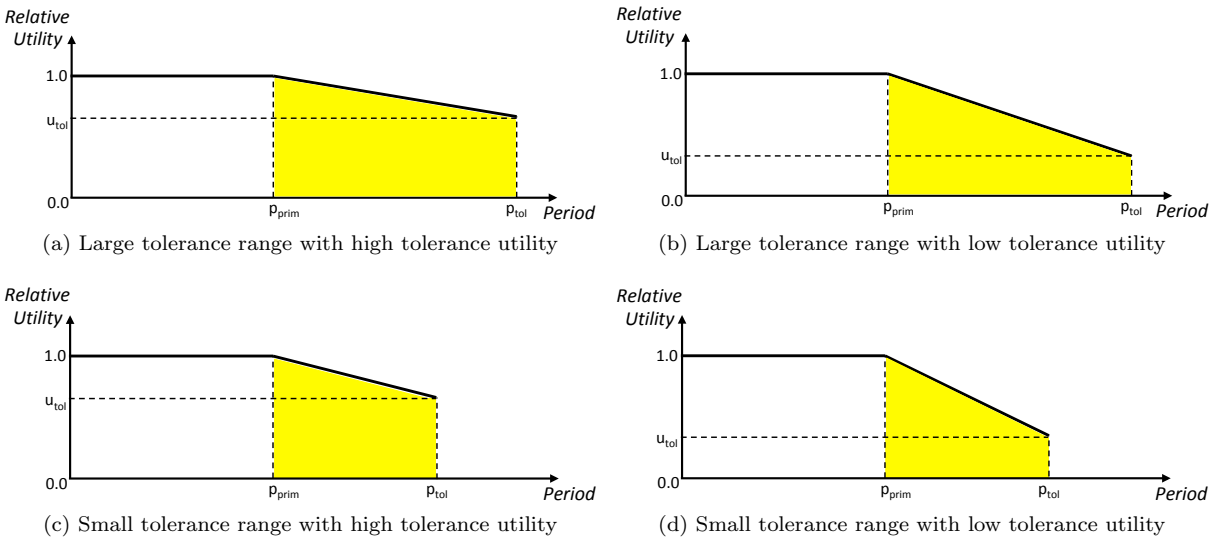
To show the effectiveness of ATMP, we compare it with a standard approach that we name *Standard Adaptive Mixed-criticality Protocol* (SAMP) in which tasks have no tolerance range. Within the SAMP approach, the tasks removal is performed only considering the load computed according to the predefined periods and no LP optimisation is performed. Once a partitioned set of tasks is assigned to a specific core, the SAMP looks for the most suitable sub set of the allocated tasks, i.e. the one with load not greater than the lm value found by binary search, by simply removing tasks with least utilisation factor among those with least criticality.

Algorithm 1: Criticality aware allocation

Input : Γ: task list sorted by criticality;
CS: list of cores;

```
1 begin
2   while Γ is not empty do
3     t_id ← getTaskWithMaxCrit(Γ);
4     c_id ← getCoreWithMinLoad(CS);
5     addTaskToCore(t_id, c_id);
6   end
7 end
```

We have created a task set consisting of twenty tasks randomly generated and then we have processed it using both ATMP and SAMP first on eight, then on five and finally on three cores. The experiment confirms that, in case of resource shortages, i.e. sudden unavailability of computing resouces, the usage of the tolerance range to appropriately optimise the tasks arrival rates allows to ATMP to de-allocate a smaller amount of tasks per core. Both approaches worked well with eight cores since no task was removed. However, ATMP showed its advantages after further reducing the number of processing elements. Because of this, we only show the performance comparison between SAMP and ATMP in case of five and three cores. Figure 3 dispays the absolute utility accrued by each individual task with the two above approaches compared with the maximum achievable utility indicated with MAX. The absolute utility of de-allocated tasks is 0. Tasks from A to H have criticality 2 while tasks from I to T have criticality 1. Figure 3.a) presents the runtime reallocation on five cores. In this case, SAMP removes five non-critical tasks while ATMP allows, via tolerance-based optimisation, to adjust the tasks periods and to keep all tasks allocated to their cores. It is worth to notice that such result is achieved also slowing down higher criticality tasks and this leads to a decrease in the overall load allocated on each core. Such results are even more emphasised in Figure 3.b) in which the number of cores available is further reduced. In this latter case, in the whole, SAMP removes thirteen tasks (two of which are highly critical ones) while ATMP removes just six



(a) Large tolerance range with high tolerance utility

(b) Large tolerance range with low tolerance utility

(c) Small tolerance range with high tolerance utility

(d) Small tolerance range with low tolerance utility

Figure 2: Service utility adaptation: tolerance range versus tolerance utility

tasks and keeps onboard all the higher criticality tasks.

Table 1 summarises the overall outcome of our experiment by showing the total relative and absolute utilities accrued within the system and the amount of tasks removed respectively by SAMP and ATMP. The total relative utility consists of the sum of the individual task utility while the total absolute utility considers also the task criticality. The more the number of cores is reduced and the more the utility gained by the tolerance-based approach increases compared with that accrued by standard one. Furthermore, our method allows to run more tasks per core when the amount of computing resources decreases.

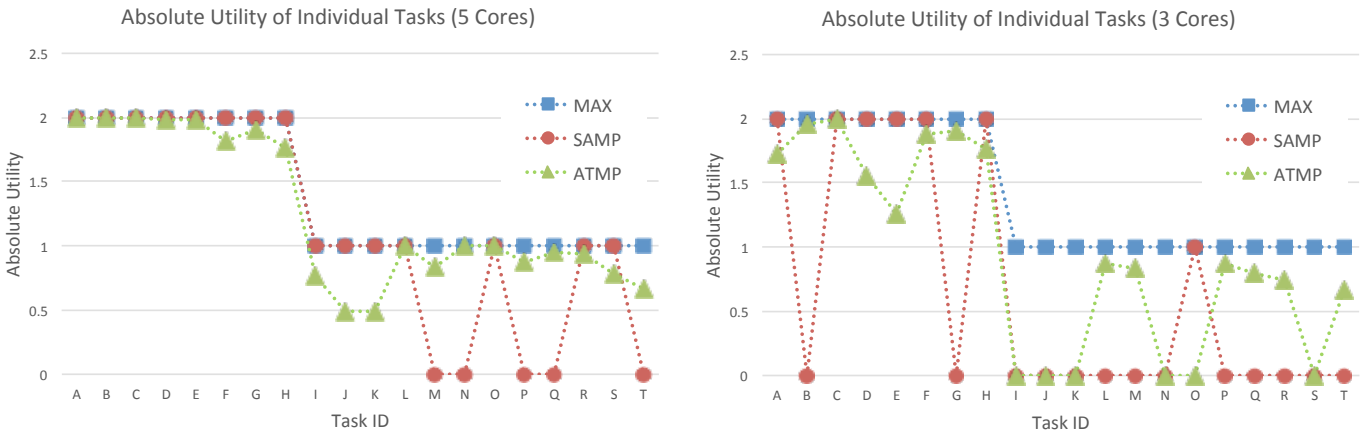
6 Safety implications of scheduling

Safety-critical systems are typically subject to two stringent correctness requirements that is necessary to consider during their design phase: a priori verification and runt-time robustness [1]. The verification determines offline whether a system will behave correctly during runtime and deals with the case when runtime behavior is compliant with its assumed model while the robustness at runtime is concerned with what happens when modeling assumptions are violated. A robust system design should ensures that performance degrades gracefully whenever resources suddenly become insufficient. In this case, a general rule is that less important system functionalities should be compromised before the most important ones.

From a safety perspective, tasks can be divided into safety-related and non-safety related, corresponding to HI and LO in case of just two criticalities. In practice, safety-related tasks can be further classified based on the extent to which they contribute to the safety of the system. These further subdivisions may be based on *Safety Integrity Levels* (SIL) [7] or similar classifications. The safety-related and non-safety related functions are required to be separated [7], since failures of non-safety related functions should not cause a dangerous failure of the safety functions.

The idea behind the mixed-criticality scheduling theory is that to construct multiple models, each of which true to a different level of assurance [1]. The successive verification of functionalities is made at the level of assurance appropriate for the specific criticality level. Such approach allows the system developer to avoid the usage of excessively conservative models to verify less critical functionalities and thus reduces the over-approximated estimates. Since both safety-related and non-safety related tasks may be run on a shared platform, as described in [2], applying mixed-criticality scheduling theory enables to design systems that are verified correct and that make a more efficient resources usage at runtime if compared with system verified using conventional schedulability analysis techniques.

The design of robust mixed-criticality scheduling protocols should allow to drop a low criticality task without impacting on the performance of any of the high criticality ones. However, this does not mean that discarding non-safety related tasks is free from consequence. These tasks may be important for non-safety reasons, e.g. to maintain sustainability, continuity of



(a) Comparison on 5 cores

(b) Comparison on 3 cores

Figure 3: Absolute utility achieved by each task with SAMP and ATMP

service or provision of capability. The advantage of this algorithm is that also non-safety related tasks, and not only safety-critical ones, can be allocated again when resourcing allows it.

6.1 Optimization and essential services

The ATMP protocol increases the system resilience against failures. In fact, if any core suddenly becomes unavailable, such method allows to de-allocate tasks according to their criticality and online adaptation capability till when the computing resources become available again. This is of particular concern for the UK critical national infrastructures supplying essential services. These services include provision of drinking water, transport of oil and gas, rail transport and medical infrastructure. Moreover, the EU has recently established a directive [23] that mandates that providers of essential services take steps to mitigate the impact of incidents which can compromise the delivery of such services, and the National Cyber Security Centre guidance [6] identifies as core principle that such essential services must be resilient, meaning that the provision of essential services should not be interrupted.

In some cases the infrastructure for the provision of essential services may also provide additional services classified as non-essential (e.g., a system monitoring and providing functionality for availability of drinking water may also be used to monitor the supply of non-potable water). Failure of essential services typically has safety implications, while temporary failure of non-essential services is unlikely to represent a safety risk. Because of this, tasks associated with the essential service may be regarded as higher criticality and those associated only with the non-essential services as lower criticality ones. In case of resource-shortages, the ATMP protocol removes first the non-essential services with worst adaptation capability and meanwhile it ensures the continuity of essential services during and after resolution of the incident. As such, our protocol could represent a potential solution for scheduling tasks on a shared platform required to comply with [23].

7 Summary and Conclusion

In this paper we have applied the *Tolerance-based Real-Time Computing Model* (TRTCM) to optimise the utility of mixed-criticality systems on multi-processor platforms. While the original mixed-criticality scheduling focuses on resource shortages due to overruns of optimistic WCET estimates, we consider also faults due to permanent unavailability of processing elements in a multi-core system.

The proposed ATMP protocol provides adaptive reconfiguration in case of resource shortages. The basic idea is to use a tolerance range of the tasks' periods to adjust the system load while at the same time optimising the system utility. Compared with the simple strategy of just abandoning tasks (SAMP) in case of resource shortage, ATMP achieves a considerably smoother degradation of service. The evaluation shows that ATMP allows to retain more tasks than SAMP, while at the same time also achieving higher overall system utilites.

Future work will include the analysis of the interplay of ATMP with a mixed-criticality protocol like the Bailout Protocol [12].

Algorithm 2: ATMP Utility Optimisation

Input : Γ: task set allocated;

Local : $lm \leftarrow 0.90$;
$lm1 \leftarrow 0.2$;
$lm2 \leftarrow 1.0$;
$lcnt \leftarrow 0$;
$\Delta \leftarrow Null$;
$bestlm \leftarrow 0.0$;
$LCNT_{MAX} \leftarrow 6$;

Output: Ω: optimised task set;

```
1  begin
2      if isSchedulable(Γ) then
3          Ω ← Γ;
4      else
5          while lcnt ≤ LCNT_MAX do
6              Δ ← Γ;
7              while (load_tol(Δ) > lm) and
                (len(Δ) > 1) do
8                  TSMIN ← getMinCritTSET(Δ);
9                  t_id ← getTaskToDrop(TSMIN);
10                 Δ ← removeTask(Δ, t_id);
11             end
12             if len(Δ) = 1 then
13                 if lm ≥ bestlm then
14                     bestlm ← lm;
15                     Ω ← Δ;
16                 end
17                 lm1 ← lm;
18                 lm ← lm + (lm2−lm)/2;
19             else if len(Δ) > 1 then
20                 prob ← genLP(Δ, lm);
21                 periods ← solveLP(prob);
22                 Δ ← getOptTSET(periods, Δ);
23                 if isSchedulable(Δ) then
24                     if lm > bestlm then
25                         bestlm ← lm;
26                         Ω ← Δ;
27                     end
28                     lm1 ← lm;
29                     lm ← lm + (lm2−lm)/2;
30                 else
31                     lm2 ← lm;
32                     lm ← lm1 + (lm−lm1)/2;
33                 end
34             lcnt ← lcnt + 1;
35         end
36     end
37     return Ω;
38 end
```

References

[1] S. Baruah. Mixed-criticality scheduling theory: Scope, promise, and limitations. *IEEE Design & Test*, 35(2):31–37, 2018.

[2] S. Baruah. Mixed-criticality scheduling theory: Scope, promise and limitations. *IEEE Design & Test*, 35(2):31–37, 2018.

[3] S. K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *Proceedings of the 2011 IEEE 32Nd Real-Time Systems Symposium*, RTSS '11, pages 34–43, Washington, DC, USA, November 2011. IEEE Computer Society.

[4] A. Burns and R. I. Davis. Mixed criticality systems - a review. Research Report V4-31/7/2014, University of York, Department of Computer Science, York, UK, July 2014.

[5] A. Burns, T. Fleming, and S. Baruah. Cyclic executives, multi-core platforms and mixed criticality applications. In *Proceedings of the 2015 27th Euromicro Conference on Real-Time Systems*, ECRTS '15, pages 3–12, Washington, DC, USA, 2015. IEEE Computer Society.

[6] N. C. S. Centre. Nis-directive: Top level objectives, 2018. https://www.ncsc.gov.uk/guidance/nis-directive-top-level-objectives.

[7] I. E. Commission. Functional safety of electrical, electronic, programmable electronic safety-related systems, 2010.

[8] R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4):35:1–35:44, October 2011.

[9] S. K. Dhall and C. L. Liu. On a real-time scheduling problem. *Operations Research*, 26(1):127–140, February 1978.

[10] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele. Scheduling of mixed-criticality applications on resource-sharing multicore systems. In *Proceedings of the Eleventh ACM International Conference on Embedded Software*, EMSOFT '13, pages 17:1–17:15, Piscataway, NJ, USA, 2013. IEEE Press.

[11] N. Guan, P. Ekberg, M. Stigge, and W. Yi. Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems. In *Proceedings of the 2011 IEEE 32Nd Real-Time Systems Symposium*, RTSS '11, pages 13–23, Washington, DC, USA, 2011. IEEE Computer Society.

[12] A. B. Iain Bate and R. I. Davis. An enhanced bailout protocol for mixed criticality embedded software. In *IEEE Transactions on Software Engineering*, volume 43, pages 298–320. IEEE, 2017.

[13] V. Izosimov and E. Levholt. Mixed criticality metric for safety-critical cyber-physical systems on multicore architectures. In *Proceedings of the 4th Workshop On Manufacturable and Dependable Multicore Architectures at Nanoscale (MEDIAN'15)*, MEDIAN '15, March 2015.

	SAMP			*ATMP*		
Cores#	*Rel. Utility*	*Abs. Utility*	*Task dropped*	*Rel. Utility*	*Abs. Utility*	*Tasks dropped*
8	20.00	28.00	0	20.00	28.00	0
5	15.00	23.00	5	17.51	25.23	0
3	7.00	13.00	13	11.79	18.81	6

Table 1: Overall comparison between *ATMP* and *SAMP*

[14] D. S. Johnsonf, J. D. Ullman, M. R. Gareyi, and R. L. Grahamii. Worst-case performance bounds for simple one-dimensional packing algorithms. 3(4), December 1974.

[15] R. Kirner. Ingredients for the specification of mixed-criticality real-time systems. In *Proc. 10th IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS'14)*, Reno, Nevada, USA, June 2014.

[16] R. Kirner. A uniform model for tolerance-based real-time computing. In *Proc. 17th IEEE Int'l Symposium on Object/Component/Service-oriented Real-Time Distributed Computing*, pages 9–16, Reno, Nevada, USA, June 2014.

[17] R. Kirner, S. Iacovelli, and M. Zolda. Optimised adaptation of mixed-criticality systems with periodic tasks on uniform multiprocessors in case of faults. In *Proc. 11th IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS'15)*, Auckland, New Zealand, April 2015.

[18] C. Lee, H. Kim, H. woo Park, S. Kim, H. Oh, and S. Ha. A task remapping technique for reliable multi-core embedded systems. In T. Givargis and A. Donlin, editors, *IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 307–316. ACM, 2010.

[19] V. Legout, M. Jan, and L. Pautet. Mixed-criticality multiprocessor real-time systems: Energy consumption vs deadline misses. In *First Workshop on Real-Time Mixed Criticality Systems (ReTiMiCS)*, pages 1–6, Taipei, Taiwan, August 2013.

[20] V. Legout, M. Jan, and L. Pautet. An off-line multiprocessor real-time scheduling algorithm to reduce static energy consumption. In *First Workshop on Highly-Reliable Power-Efficient Embedded Designs (HARSH)*, pages 7–12, Shenzhen, China, February 2013.

[21] B. Motruk, J. Diemer, R. Buchty, R. Ernst, and M. Berekovic. Idamc: A many-core platform with run-time monitoring for mixed-criticality. In *IEEE 14th International Symposium on High-Assurance Systems Engineering (HASE)*, pages 24–31. IEEE Computer Society, October 2012.

[22] R. Obermaisser and D. Weber. Architectures for mixed-criticality systems based on networked multi-core chips. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation, ETFA 2014, Barcelona, Spain, September 16-19, 2014*, pages 1–10, 2014.

[23] E. Parliament and the Council of the European Union. Directive (eu) 2016/1148 concerning measures for a high common level of security of network and information systems across the union, 2016.

[24] H. Su, D. Zhu, and D. Mossé. Scheduling algorithms for elastic mixed-criticality tasks in multicore systems. In *2013 IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications*. IEEE, August 2013.

[25] A. Thekkilakattil, A. Burns, R. Dobrin, and S. Punnekkat. Mixed criticality systems: Beyond transient faults. In *Proceedings of the third International Workshop on Mixed Criticality Systems (WMC)*, San Antonio, Texas, USA, December 2015.

[26] A. Thekkilakattil, R. Dobrin, and S. Punnekkat. Mixed criticality scheduling in fault-tolerant distributed real-time systems. In *2014 International Conference on Embedded Systems (ICES)*, pages 92–97, July 2014.

[27] S. Tobuschat, P. Axer, R. Ernst, and J. Diemer. Idamc: A noc for mixed criticality systems. In *IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2013, Taipei, Taiwan, August 19-21, 2013*, pages 149–156, August 2013.

[28] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *28th IEEE Real-Time System Symposium (RTSS 2007)*, December 2007.

[29] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proc. 28th IEEE International Real-Time Systems Symposium (RTSS'07)*, pages 239–243, Dec. 2007.