# Controlling High-Performance Platform Uncertainties with Timing Diversity

Robin Hapka, Anika Christmann, Rolf Ernst
{hapka,christmann,ernst}@ida.ing.tu-bs.de
*Institute of Computer and Network Engineering*
*Technische Universität Braunschweig*
Germany

***Abstract*—Autonomous mobile systems combine high performance requirements with safety criticality. High performance hardware/software architectures, however, expose a far more complex runtime behavior than traditional microcontroller architectures. Such high-performance architectures challenge traditional worst-case design that assumes a formally analyzable or at least deterministic worst-case response time (WCRT) that can be reasonably bounded. However, such architectures expose rare but substantial worst-case outliers, which are not only caused by the application itself, but also by the many dynamic influences of software architecture and platform control. Probabilistic methods can capture such outliers, but are only effective, if the outlier probability is sufficiently low and if the methods cover dynamic platform timing. As a main contribution, this paper exploits platform induced timing variety rather than trying to mitigate it. Assuming the typical redundant dual modular redundancy (DMR) implementation that is deployed in safety-critical systems, it introduces the concept of Timing Diversity, where rare outliers in one of the two channels are masked by the other channel with a sufficiently high probability. The paper uses a convolutional neural network (CNN) example in different parameter settings running on Linux operated multi-core platform with typical dynamic control to investigate the proposed concept. The experiments demonstrate the potential of Timing Diversity in leading to substantially higher reliability. Alternatively, the approach permits a reduction of the system WCRT at the same reliability level.**

***Index Terms*—high-performance platform, execution time uncertainties, convolutional neural network, dual modular redundancy, Timing Diversity**

## I. Introduction

Current and emerging high-performance embedded real-time platforms must keep up with the ever-increasing performance demands of autonomous systems. Though commercial off-the-shelf (COTS) multi- and many-core processors can provide such performance, they exhibit much slower timing in the worst case. This critical issue is caused by the multi-level cache hierarchy, the stateful behavior of large memories, especially DRAM, internal network congestion, or power control, to just name few effects. Still, the worst-case execution time (WCET) and worst-case response time can be upper-bounded conservatively [1], but will likely exceed the given performance capabilities, due to its pessimistic assumptions. Invasive mechanisms to actively increase time predictability, e.g. by memory access shaping (e.g. [2]) affect performance and will hardly be accepted in performance demanding applications.

Additionally, the complexity and number of applications running on an autonomous system platform is also increasing and the industry is forced to deal with this. To master the complexity, the industry is tempted to use Linux instead of a classic real-time operating system (RTOS). The use of Linux comes with many advantages. First, Linux enables high productivity as required for the emerging agile processes that expect updatability after deployment. Secondly, many hardware devices such as GPUs and software libraries can be used out-of-the-box. Finally, Linux supports parallel programming on thread and core level. However, the Linux software architecture with its data management, call structure, signaling and scheduling leads to many interleaving timing effects that are hard to control and complicate the WCRT bound problem even further. Though Linux certainly adds to the WCRT dynamics, it is unlikely that an RTOS with the requested functionality running on high-performance platforms will substantially reduce the WCRT effects.

In consequence, there seems to be no other way than either to accept the reduced performance or to derive the WCRT of an application in a sufficiently long test sequence to support the required level of deadline guarantees (e.g. the maximum probability of misses). Given the reliability targets of safety critical or high availability systems, this seems to be a very costly and time-consuming solution that will hardly be accepted in practice.

In this paper, we will propose another approach to the WCRT challenge that resorts to an established technique from fault-tolerant systems design, i.e. modular redundancy. In systems with higher criticality, modularity is mandatory to raise the reliability of a system above that of their components. A typical example is automated driving, where DMR is used to keep the system operational if one module fails (“fail operational”). However, modular redundancy is only effective if modules are sufficiently independent, such that a common failure is highly unlikely. That is typically the case for properly separated hardware modules, such that error calculation may assume a single error model. But, how could that be the case for execution timing if both modules run the same software with the same input data on an identical execution platform? That should almost be excluded under the usual assumption that, if software execution is not predictable, then it is at least repeatable.

*Contribution:* The first contribution of this paper is to demonstrate by measurement that the timing repeatability assumption does not generally hold for COTS high performance hardware/software architectures. Instead, response times may vary in consecutive executions under the same input conditions. The measurements for several examples show that the variance particularly affects the positions of timing outliers. The seemingly negative result on execution timing can be exploited in a way that, to our knowledge, has never been proposed before (the concept is outlined in paper [3] and the WIP paper [4] written by the author of this paper). This exploitation is the second contribution. Instead of trying to improve predictability or mitigate the outliers at the cost of performance, we exploit concurrent executions on identical modules in such a way that they mask each other when executed in modular redundancy. For a more formal treatment, we extend the existing single independent error model of DMR to single independent deadline violations and derive the necessary check operations and their timing. We finally provide experimental evidence that the resulting DMR setup works in practice.

The remainder of the paper begins with an overview of established methods and related work. We then present our approach in detail and describe a machine learning use case inspired from the avionics industry exploiting existing DMR. At last, we evaluate the results obtained from the use case and finish with a conclusion.

## II. Related Work

Handling multi-core processor uncertainties is a challenge in itself without even considering the entire platform. In this section, we take avionics as an example, because of the strict requirements and related methods that are found in this domain giving good insight in the problem space. Similar challenges arise in other safety-related domains, such as automotive and industrial electronics. We have recognized the trend in industry to integrate highly parallelized functions in autonomous systems. Therefore, we focus on: (1) using highly parallelized applications such as neural networks, (2) which fully exploit the performance capabilities of a COTS multi-core processor. We first give a brief overview of the many different approaches and then check whether they are suitable for our requirements. We conclude with a discussion with respect to platform uncertainties.

*1) Architecture and Mechanism:* Several lines of research focus on approaches and mechanisms to ensure safety and timing guarantees on multi-core architectures. As an example, [5] identifies interference and its impact on determinism through benchmarking to ensure safety guarantees. By integrating mixed-criticality applications on a multi-core platform that perform multiple memory accesses in parallel, the authors aimed to stress the interconnect, system level caches and main memory. In [6] an interference-sensitive WCRT was introduced which extend modern timing analysis techniques. It takes into account the capturing of resource utilization and the calculation of interference delays. Furthermore, it was extended by runtime monitoring and shaping to enforce timing guarantees. This approach focuses on integrating mixed critical applications. Reducing dependencies between them might be at least possible if only the critical application is running. In our case, we dedicate the platform to a single challenging application. Consequently, performance guarantees at the cost of less critical applications is not possible.

*2) Deterministic Execution Model:* In [7] a scheduler is implemented to define rules to constrain the unpredictable behavior of multi-cores. Since the scheduler is intended to lead to deterministic timing behavior, this branch of research is commonly referred to as deterministic execution model. In [7] propose a slice execution model. There, the scheduler distinguishes between two types of slices: execution slice and communication slices, and only in a communication slice access to shared resources is allowed. A performance-driven application is slowed down if it gets only access to shared resources in a certain phase or slice. Our performance requirements are therefore not met.

*3) Memory Access Control:* Since memory access causes interference, a different approach is proposed, namely memory access control. In [2] a software-based memory throttling mechanism is introduced. Except of the so-called critical core, all memory access of the cores is controlled. While the timing requirements of a safety-critical task are met, the overall performance of the system suffers [8]. Therefore, the work is extended by a fine-grained memory bandwidth system called MemGuard [8]. It is based on a static memory bandwidth throttling. The main challenge for such core centric approaches is the very complex and stateful behavior of current memories, such as DDR4, leading to strong variations in timing [9] and large worst-case outliers [10]. Follow-up work, e.g. [11], therefore, addresses stateful memory control, including the use of COTS IP cores and memory functionality, but again for integration in mixed-critical applications.

*4) Real-Time Capable Design:* The research projects MERASA [12] and parMERASA [13] focus on simplifying the WCET analysis of high performance architecture by developing a deterministic multi-core architecture. MERASA's hardware architecture includes an interference-aware bus arbiter, a dynamically partitioned cache and an analyzable real-time memory controller. It also isolates tasks at core level to minimize inter-task interference. Due to shared resources such as shared memory and bus-based techniques, the design is limited to four to eight cores. This approach provides isolation and determinism for the applications rather than focusing on maximum performance and minimum latency. Therefore, it violates our performance requirements. Moreover, the high initial cost of developing new hardware, the probably low quantities and the performance drawbacks make it unlikely that such custom hardware will be manufactured, except for research purposes.

*5) Probabilistic Timing Analysis:* Another field of research is the probabilistic timing analysis (PTA). In contrast to the traditional timing analysis, repeated executions are analyzed and a probability distribution is derived instead of a scalar

value [14]. The PTA can be classified into five main fields [14]. One of them is the measurement-based probabilistic timing analysis (MBPTA). The idea of MBPTA is using extreme value theory to estimate a probabilistic worst-case response time (pWCRT) based on observed executions. The methods of PTA were used within the PROARTIS project [15]. The central hypothesis of the project is that multi-core processors and software features enable a truly randomized timing behavior. Based on this assumption, it can be verified that the probability of response time outliers is negligible. In the successor project PROXIMA, the objective was to provide a complete toolchain enabling PTA methods on multi-core and many-core processors [16].

Similar to the MBPTA, we measured the response times of our application. But MBPTA computes an exceedance function, which they use to specify a likelihood beyond which timing outliers are negligible. Similar to the other approaches, they focus on calculating a pWCRT that is unlikely to be exceeded.

*6) Real-time based Artificial Intelligence (AI):* With respect to AI application, the work of [17] is very instructive. It is inspired by an automotive industry case study and focused on CNN applications in computer vision for autonomous vehicles. By using a state-of-the-art automotive platform, the authors aim to demonstrate that pipelining a CNN for multiple camera frames on a combination of a multi-core CPU and two GPUs can significantly increase the maximum frame rate. The approach focuses on system throughput and omits 5% longest observed response times when calculating latency. The authors argue that Linux produces such timing outliers that should not be considered. However, a frame drop rate of 5% corresponds to at least several missed frames per second (even higher if bursts are possible), which is a high level for vision applications and could possibly lead to late or incorrect object detection and motion estimation.

Another approach is the work of [18]. The author focus on analyzing and minimizing end-to-end delay of real-time object detection for autonomous driving. Right at the beginning, they investigate the end-to-end delays as well as the cycle time of different deep neural networks running on different GPUs, respectively. They observe that the performance varies significantly depending on the deep neural network and hardware combination. In addition, they note that there is an imbalance between the cycle time and the delay, which is sometimes 6 times greater. One of the reasons for the time lags is the fact that the GPU and CPU run simultaneously and thus compete for the shared memory bandwidth. To minimize memory bandwidth contention between GPU and CPU, the authors applied synchronized executions between GPU and CPU. This leads to a temporal isolation and the loss rate can be reduced to 1%.

*7) Discussion:* The concurrent execution of multiple applications on different cores can lead to accesses to shared resources that can negatively impact the runtime behavior of a single application. All related works agree that this behavior is undesirable for safety-critical applications. Therefore, they aim to guarantee the timing of such applications by introducing mechanisms or novel hardware designs to determine tight upper bounds on WCRT. Additionally, most approaches aimed to integrate mixed-criticality application on a single hardware platform. With different restrictions it is possible to guarantee timing behavior of safety-critical applications. But this is usually achieved with performance drawbacks of the entire system. However, we only have one highly parallelized function that also uses the entire performance of the multi-core. Our application have no performance reserves that we can use and there are also no non-critical applications that can be turned off.

In contrast, the MBPTA approach assumes that the random behavior of hardware platforms enables the extreme value theory to estimate the exceedance function of the pWCET distribution. Finally, this approach also aims to obtain tight upper bounds on the WCRT but takes into account the likelihood of large response times. In turn the WCRT (independent whether it is estimated or formally calculated) does highly dependent on input data, execution path, and the internal state of the hardware. Consequently, only considering the underlying hardware is insufficient, instead the entire platform must also be taken into account.

With respect to the use of AI in real-time critical systems, the focus is on the short latency of detection pipeline. Both [17] work and [18] work assume loss rates of 5% and 1%, respectively, which results in the exceedance of deadline. Deadline misses are, in our opinion, intolerable, as they make it difficult to certify such systems. To the best of your knowledge, no proposed work meets our stringent deadline in AI-based object detection and recognition Therefore, the focus on meeting deadlines seems to be an open problem.

## III. Use Case

In industry, the trend is toward integrating highly parallelized functions in autonomous systems. This trend results in the following requirements for our use case: (1) it must be a highly parallelized application, (2) which fully exploits the performance of a COTS multi-core processor. In addition, (3) the application must be data-independent and have a (4) sufficiently long execution time to be able to exclude short-time effects.

We have been inspired by an important area for high performance architectures, namely avionic AI applications. Recently, the European Union Aviation Safety Agency (EASA) have published several documents addressing the deployment of avionic AI application [19] [20] [21] [22] and they expect the first certification of pilot assistance systems in 2025 [20, p. 12]. This corresponds to a highly parallelized application utilizing the entire performance abilities of modern multi- and many-core processors, and therefore the requirements of EASA for future avionics systems are very similar to ours. A representative example that benefits from neural networks is the visual landing guidance [19, Chapter 4].

Instead of visual landing guidance, our application detects objects in input images and classifies each object found into

one of a thousand predefined classes. For this purpose, the input image is received over a small network using UDP/IP in combination with multicast. Afterwards, the image frame is preprocessed according to the needs of the underlying neural network. This includes resizing and cropping the input frame as well as subtracting the mean value. Subsequently, the neural network is executed and its output vector is post-processed. Afterwards, the program iterates over each entry in the output vector, resolves the objects class ID into a class name and puts a bounding box, together with the corresponding class name on the image. An entry might be skipped, if the confidence value is not sufficiently high. In summary, our safety-critical application can be divided into three stages: 1) preprocessing, 2) neural network execution and 3) post-processing.

We examined different neural networks, which are all based on the single shot multibox detection (SSD) technique developed by [23]. Though, some neural networks offer a more accurate detection than SSD, these are unfeasible for embedded systems, since they require high-end hardware acceleration [23]. Namely, we studied MobileNets [24] in the version V2, V3 small and V3 large [25]. These networks are designed for perception tasks and especially for embedded and mobile devices. Therefore, lower computational costs are required. As to achieve that, the author of MobileNets enhanced the SSD technique by their so called SSDLite approach. This reduces the computational costs, while improving accuracy slightly, too [24]. Models of the pretrained neural networks are freely available. We deployed these by using the deep neural network module provided by OpenCV (in version 4.5.5). This allows the application to be written in C/C++, purely.

We decided to focus on CPUs, as a first step. This decision is motivated by the fact that even in a system with hardware acceleration, still multi-core processor and Linux are deployed, as well [17], [18]. So, the problem statement remains relevant under that circumstances. Finally, the class of neural network applications is predestined for our use case, since the execution path remains nearly constant despite varying input data. Hence, we executed all stages of our sample application on a multi-core CPU and used resource-friendly neural networks and low-resolution images.

## IV. Initial Experiment: Impact of Platform Dynamics

Let's take a look at how platform dynamics caused by COTS multi- and many-core processors combined with Linux affects our use case. For this purpose, we executed our AI use case 50,000 times, each time with the same input data, and measured the response time of our application. We then repeated the measurement a second time on the same hardware and with the same input data.

In Fig. 1 the result of the two measurement series is shown. We denoted those two measurement series as dataset 1 (blue) and 2 (green), respectively. The y-axis shows the response time for a corresponding execution on the x-axis. In regard to the response time behavior, it can be seen that dataset 1 and dataset 2 are very similar on average, but not identical. Both oscillate between 460 ms and 510 ms. In total, there are six timing outliers: two in dataset 1 and four in dataset 2. Analyzing the timing outliers in the respective dataset, it can be seen that the height of the outliers differs significantly. For example, the response times of the first two outliers in dataset 2 are greater than 700 ms, while the third (run 27,000) takes only 560 ms. Comparing the timing outliers among the two datasets, it is also obvious that the position is different.

Though we used the same input data in each iteration and tried to conduct the measurements as identical as possible, the response times within a single dataset differed significantly. This means that the response times vary in consecutive executions under the same input conditions and therefore, the response time behavior cannot be repeated. With this experiment we have thus shown, that the timing repeatability assumption does not generally hold for COTS high performance hardware/software architectures. As a consequence, there seems to be no other way than either to accept the reduced performance or to derive the WCRT of the application in a sufficiently long test sequence to support the required level of deadline guarantees.

## V. Timing Diversity

The lack of repeatability can be exploited, to tackle the challenge of meeting deadlines. Instead of predicting or limiting response time outliers with complex techniques, we handle these by simple means of redundancy. To use modular redundancy, it is necessary that the modules are sufficiently independent, such that a common failure is highly unlikely. The measurements from Fig. 1 indicates exactly such a sufficiently independent behavior due to the unrepeatable response times of timing outliers. This observation enables us to use modular redundancy for COTS multi- and many-core processors.

By applying modular redundancy, at least two redundant units are in operation, usually referred to as dual modular redundancy. This technique is commonly used to detect hardware (HW) errors and software (SW) errors, additionally we extend it to mask timing outliers. The concept of masking timing outliers in a system with existing redundancy is shown in Fig. 2. Two redundant channels (Ch A and Ch B), synonymous with module, execute equivalent software on equivalent hardware under identical deadline constraints. At the end of processing, the evaluation of the redundant results is performed by a hardware or software Comparator (Comp) releasing the correct result or reporting errors [26, Chapter 4.3.1 and Chapter 7].

The detection of HW errors, SW errors or timing errors underlies the *single-error model*, as is common in DMR. Fig. 2 represents the case when on Ch A a HW or SW error has occurred. According to the single-error model, Ch B has no error, and Ch B returns the correct result before the deadline. Three cases are possible for the result on Ch A: *(a)* a correct result i.e. the error is not effective, *(b)* an incorrect result or *(c)* there is no result at all (deadline miss). In case of a correct or incorrect result, the result of either Ch A or Ch B can be taken (see Fig. 2 (*a*)/(*b*) take any). If the subsequent

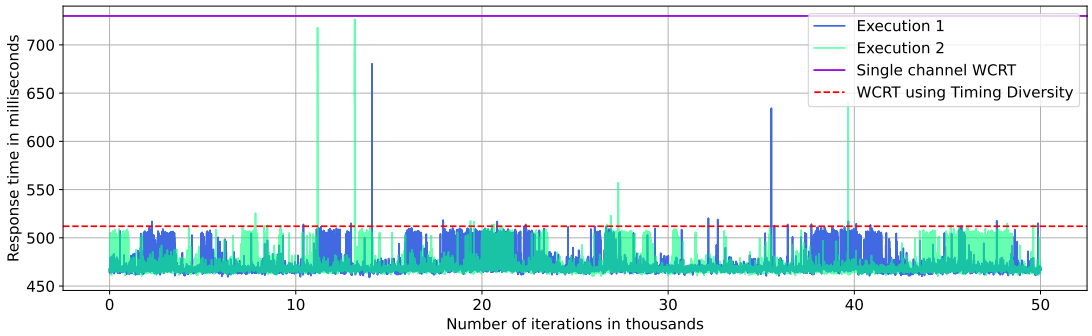


Fig. 1: Response times of two series of measurements

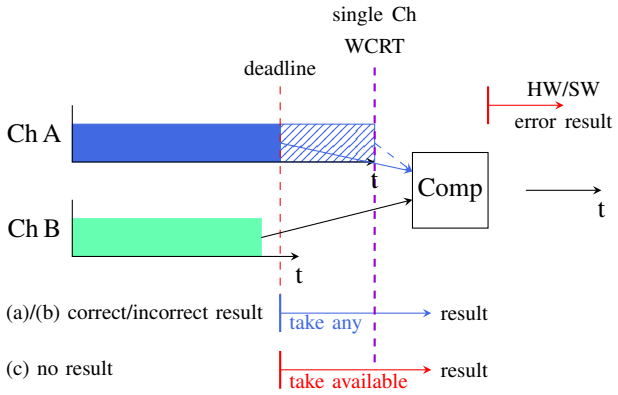


Fig. 2: Extension of the DMR function to mask timing outliers

comparison process detects an incorrect result, the safety layer is entered, otherwise the processing continues. In case of no result (Fig. 2 (*c*)) DMR mechanisms cannot determine whether the deadline miss is due to an irrecoverable HW/SW error or a recoverable timing error.

This requires an extension of the DMR function. Instead of canceling Ch A's task at the (missed) deadline, it continues to execute until its single channel WCRT (see Fig. 2 shaded area). Since Ch B provides the result in time, the available is taken (Fig. 2 (*c*)). Ch A will provide a correct, an incorrect or no results at single channel WCRT. In case of a correct result, Ch A had a recoverable timing error and possibly a non-effective HW or SW error. It is possible to proceed without further action. In case of an incorrect or no result, the deadline miss was due to a HW or SW error or an "excessive" timing error. The comparison process detects that and enters the existing safety layer.

If only a timing error occurs on Ch A and no HW or SW error, the single-error model still holds that Ch B provides the result before the deadline. The result on Ch A could be correct, but not in time. This means at deadline, Ch A provides no result and it must be waited until the single channel WCRT (as shown in Fig. 2) to obtain information about the cause of failure.

The bottom line is, applying the single-error model, at least one result is available at the specified deadline. Determining whether a deadline miss is caused by recoverable a timing or an irrecoverable HW or SW error, is only possible at single channel WCRT. We call this approach **Timing Diversity**.

## VI. Hardware Setup

As hardware for our experimental evaluation we deployed three single board computers (SBCs) with 2GB of RAM. Two executes our AI application, the other one transmits video frames via UDP/IP to the other two, periodically. All SBCs are based on a quad-core processor featuring four ARM Cortex-A72 cores. Since, we are not able to give any hardware design assurance, we decided to use a multi-core design with a lot of in-service experience. This core was first launched in 2016 and was designed as the new high-end processor core replacing the ARM Cortex-A57. As such, it is widely distributed and is used in chips from many manufactures like Broadcom, HiSilicon, MediaTek, NXP, Marvell, Rockchip, Qualcomm, Texas Instruments and Xilinx.

On top of our SBCs Linux runs as an operating system (in kernel version 5.10.63). In contrast to RTOS, Linux is optimized on average performance and throughput, instead of determinism and upper bounding the response time. In combination with the best-effort scheduling of Linux background tasks might impact safety-critical applications, eventually, if not otherwise configured. Therefore, a safety-critical application suffers from interference introduced directly by the underlying multi-core hardware and interference caused by the operating system. With Timing Diversity, we offer a mechanism to deal with platform uncertainties regardless whether caused by hardware or software. Furthermore, Timing Diversity is not restricted to Linux and can be used with any other general-purpose or real-time operating system.

## VII. Results and Evaluation

In this section, we evaluate the benefits of Timing Diversity through experiments. For this purpose, we executed our safety-critical application deploying different versions of MobileNet a 100,000 times each, simultaneously on both platforms and measured the overall response times. In this context, the total response time starts after the frame has been received, with the beginning of the preprocessing, and ends when the post-processing finished. Note that network transmission

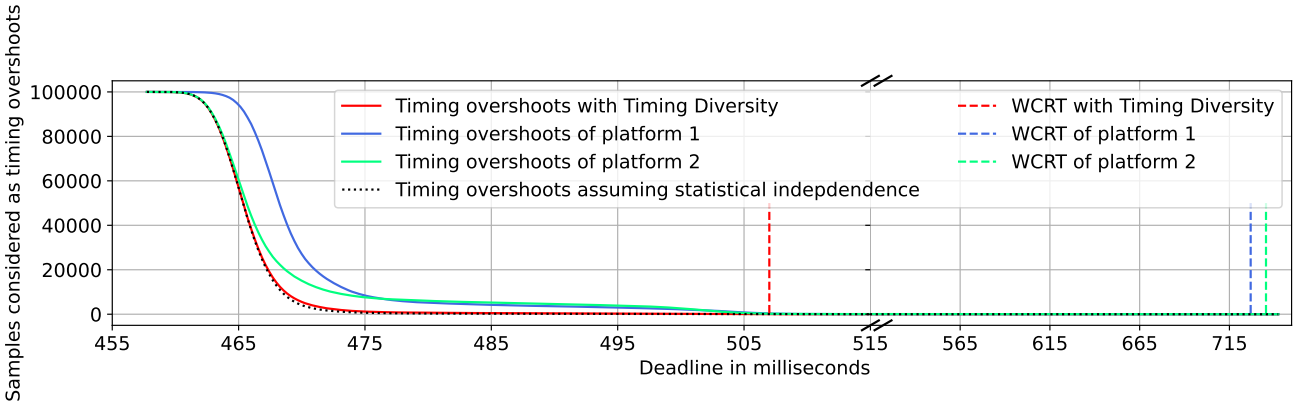


Fig. 3: Deadline-distribution showing overall response times of MobileNet V3 large at 600MHz clock speed.

jitter may lead to small shifts in the beginning of execution between platforms. Also, the neural network runs once with a completely black image as input before the first measurement begins to avoid the measurement of cache warm-up.

First, we take up the observation from Section IV. In order to use DMR and thus Timing Diversity, the timing outliers must be sufficiently independent. Therefore, we measured the deadline-distribution, which is shown in Fig. 3. There, an arbitrary deadline is assumed and those executions are counted, which missed the specified deadline. The number of samples exceeding the deadline is plotted on the y-axis. The x-axis offers a wide range of deadlines starting with the best case and ending with the worst-case response time. The colors of the different curves correspond to the first and second platform as well as to the Timing Diversity approach and are equal to those used in the previous figure. Additionally, the dotted, black line indicates the number of timing overshoots per deadline under the assumption of statistical independence. I.e., the measured probabilities of timing overshoots of both channels are multiplied for each deadline and converted into absolute values. Furthermore, the dashed, vertical lines marks the observed WCRT of each curve according to their specific color. Moreover, two diagonal lines indicate a break of scaling of the x-axis.

Fig. 3 allows estimating the difference between Timing Diversity and an ideal system with modular redundancy under the assumption of statistical independence of both channels. The solid, red curve and the dotted, black line are well aligned for wide ranges of the x-axis, showing that the use of redundancy in Timing Diversity is almost ideal, at least for the investigated use cases.

Because, Platform 2 is usually faster in the average case than the other platform, the whole system fails if Platform 2 fails. But for redundancy to work, it requires both redundant units to have a chance of correcting the other. Otherwise, the redundant unit has no use. This points out the importance of equally fast platforms and explains the overlapping curves of Timing Diversity and Platform 2 in the beginning. Though, for deadlines above 470 ms the situation changes and both platforms become about the same speed.

An alternative representation of the deadline-distribution is the violin plot in Fig. 4, which shows the response time distribution in three columns. Fig. 4 is based on the same data as the violin plot (Fig. 3). The left and middle column referring to Platform 1 and 2, while the right displays the distribution with Timing Diversity. Here, for every execution the lower response time of both platforms is considered, since only one result needs to be available in time. The y-axis is shared among all columns and describes the response time in milliseconds. Each column consists of a vertical bar indicating the range of response time data. At both ends a horizontal line marks the minimal and maximal value. The colored shade around the bar is called density plot and reports the frequency of the corresponding response time value on the y-axis.

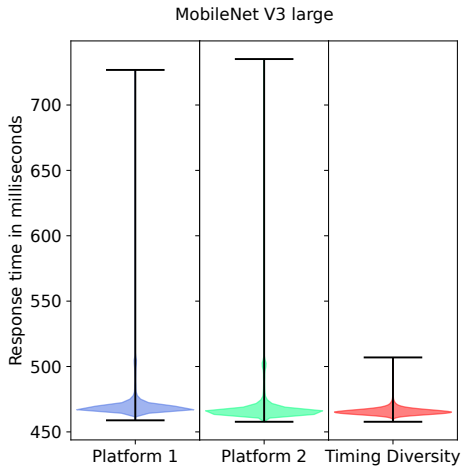


Fig. 4: Violin plot showing the response times of MobileNet V3 large at 600MHz clock speed.

The violin plot in Fig. 4 shows that the best case response time is about 458 ms for all columns. The observed average case response time is at 467 ms for Platform 1, 466 ms for the redundant platform and 464 ms in combination with Timing Diversity. Additionally, a local maximum with respect

to frequency is shown in the density plot around 503 ms for the first platform or 502 ms for the second, respectively. Although the impression might be that Platform 2 is slightly faster on average than the first, in the worst case it is the other way around. Platform 2 needs 735 ms, while Platform 1 finishes after 727 ms, at worst. These variations are in the magnitude of a single percentage, usually less, and can be dismissed as a coincidence. Both platforms feature a tail distribution, whereas the rarely observed WCRT is noticeably larger than the average response time. Some authors even measured slowdowns about a factor of about 5 [5] [27]. Although, we observed a WCRT of only about 55% larger than the average case, it displays the problem of multi-core processors, clearly: very large, but rare response time outliers restrict the real-time performance.

Fig. 4 shows that applying Timing Diversity reduces the observed WCRT to 507 ms. This corresponds to a reduction of 31% in comparison with Platform 2. This shows that if deadline misses occur sufficiently rare, Timing Diversity can be exploited to reduce the deadline.

As a best practice means, a relative safety margin would be put on top of the observed WCRT according to [28]. For example, consider 25% as our safety margin, then Platform 2 would result in 919 ms and Timing Diversity in 634 ms. Since we calculated with a relative margin, the reduction remains by 31%. In summary, Timing Diversity does not change or limit the applicability of additional safety means and in systems with existing redundancy, Timing Diversity comes without additional costs.

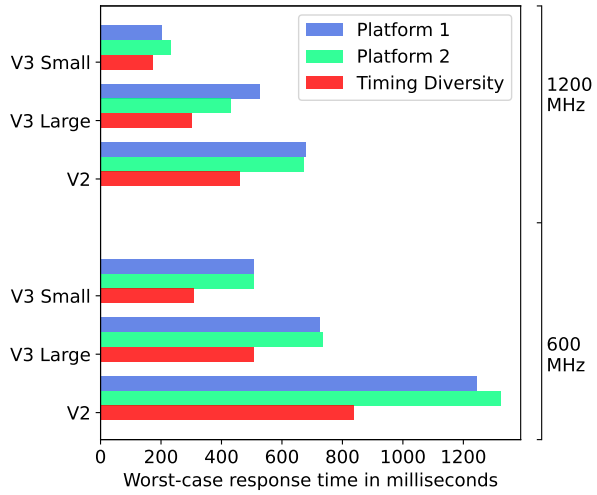


Fig. 5: Bar plot displaying the observed WCRT of different neural networks using 100,000 measurements each.

Additional to MobileNet V3 large, we repeated the measurements for MobileNet V2 and V3 small, using different clock frequency. The response time distributions of each are similar to the one of V3 large presented in Fig. 4 and are omitted for clarity. Instead, the observed WCRT of each neural network and frequency is shown in Fig. 5. The achieved reduction of the observed WCRT depends on multiple factors and varies from 26.6% up to 42.8%. Though, a general statement about the reduction is difficult, it should be noted that the WCRT of Timing Diversity is always smaller than of the faster single platform alone.

We conclude with an outlook on how the result of this paper can be applied in different verification methods:

*1) Method Test:* The first approach is to use the measurements as a system test for timing errors. In this case, we observe no deadline violation (Fig. 1) in the interval of 50000 executions corresponding to 7h of execution time. Such a test will be sufficient for lower criticalities, but a result for the limited time interval cannot easily be extended to conclusions concerning millions of hours of operation, as requested for higher criticalities.

*2) Method Exploiting Test Statistics:* This method uses the same measurements, but applies standard methods to support the assumption of statistical independence of both channels. If the independence can be supported, the compound probability of deadlines misses is taken. To compute a compound probability, the method requires violation probabilities greater than 0 on the individual channels, as in the case of the red deadline. While the method provides results for significantly higher reliability levels than method 1, application to the highest criticalities requires extremely long measurements.

*3) Method Exploiting MBPTA:* This method uses the MBPTA as proposed in the literature [14, Section 2.3]. It does not require deadline misses, such that we can evaluate reliability for longer deadlines, such as the purple deadline in Fig. 1. In this case, the entire redundant system is considered as a single system. Using the approach proposed in the literature, the MBPTA based method can be used for any level of criticality.

## VIII. Conclusion

We started from the observation that in high-performance embedded platforms the positions of rare response time outliers between consecutive series of measurements vary and are generally not repeatable. While this is a hard challenge for worst-case design and analysis, we proposed to exploit this seemingly unfavorable result to establish Timing Diversity in modular redundancy architectures. Modular redundancy is already available in platforms for critical functions, where it provides fault tolerance for fail-operational behavior. We investigated Timing Diversity for several machine learning algorithms on a physical dual-modular redundant (DMR) platform and showed that, in all cases, the individual rare timing outliers were masked by diversity, supporting the assumption of a single-error model. Such masking leads to substantially shorter worst-case system response times or to far higher system reliability than the individual model. Finally, we proposed three methods for verification of systems with Timing Diversity, which differ in result quality and required independence properties. They can be applied to different levels of required reliability. At the current stage, the approach assumes sufficient recovery time between task activations, such that timing outliers do not delay subsequent activations. As

future work, we plan to investigate Timing Diversity under periodic activations shorter than outlier response times to extend timing diversity to higher data rates and pipelined systems.

## ACKNOWLEDGMENT

This work was funded by the German Federal Ministry of Economic Affairs and Energy (BMWi) within the Many-core Avionics Design, Architecture, Modeling and Simulation (MC-ADAMS) project, funding number 20E1920B. We would like to thank the other members of the MC-ADAMS project for their support in our research.

## REFERENCES

[1] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem—overview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 3, may 2008.

[2] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "Memory access control in multiprocessor for real-time systems with mixed criticality," in *2012 24th Euromicro Conference on Real-Time Systems*. IEEE, 2012.

[3] A. Christmann, A. Kostrzewa, R. Ernst, M. Rockschies, M. Halle, F. Thielecke, A. Peuker, A. Kuzolap, M. Steen, P. Hecker, K.-F. Nessitt, and S. Saidi, "Integrating multi-/many-cores in avionics: Open issues and future concepts," in *2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*. IEEE, 2021, pp. 1–8.

[4] M. Möstl, R. Hapka, A. Christmann, and R. Ernst, "Work-in-progress: Timing diversity as a protective mechanism," in *2021 International Conference on Embedded Software (EMSOFT)*. IEEE, 2021, pp. 29–30.

[5] J. Nowotsch and M. Paulitsch, "Leveraging multi-core computing architectures in avionics," in *2012 Ninth European Dependable Computing Conference*, 2012, pp. 132–143.

[6] J. Nowotsch, M. Paulitsch, D. Buhler, H. Theiling, S. Wegener, and M. Schmidt, "Multi-core interference-sensitive wcet analysis leveraging runtime resource capacity enforcement," in *2014 26th Euromicro Conference on Real-Time Systems*. IEEE, 2014.

[7] F. Boniol, H. Cassé, E. Noulard, and C. Pagetti, "Deterministic execution model on cots hardware," in *Proceedings of the 25th International Conference on Architecture 700 of Computing Systems*. Springer, Berlin, Heidelberg, 2012, pp. 98–110.

[8] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms," in *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2013.

[9] Shang Li, Dhiraj Reddy, Bruce Jacob, "A performance & power comparison of modern high-speed dram architectures," in *Proceedings of the International Symposium on Memory Systems*, New York, NY, USA, 2018.

[10] L. Ecco and R. Ernst, "Tackling the bus turnaround overhead in real-time sdram controllers," *IEEE Transactions on Computers*, vol. 66, no. 11, pp. 1961–1974, 2017.

[11] M. Hassan and R. Pellizzoni, "Bounding dram interference in cots heterogeneous mpsocs for mixed criticality systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2323–2336, 2018.

[12] T. Ungerer and et ali., "Merasa: Multicore execution of hard real-time applications supporting analyzability," *IEEE Micro*, vol. 30, no. 5, pp. 66–75, 2010.

[13] ——, "Parallelizing industrial hard real-time applications for the parmerasa multicore," *ACM Transactions on Embedded Computing Systems*, vol. 15, no. 3, pp. 1–27, 2016.

[14] R. I. Davis and L. Cucu-Grosjean, "A survey of probabilistic timing analysis techniques for real-time systems: 03:1-03:60 pages / leibniz transactions on embedded systems, vol 6, no 1 (2019)," *LITES: Leibniz Transactions on Embedded Systems*, pp. 1–60, 2019.

[15] F. J. Cazorla, E. Quiñones, T. Vardanega, L. Cucu, B. Triquet, G. Bernat, E. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo, and D. Maxim, "Proartis: Probabilistically analyzable real-time systems," *ACM Transactions on Embedded Computing Systems*, vol. 12, no. 2s, pp. 1–26, 2013.

[16] L. Kosmidis, E. Quiñones, J. Abella, T. Vardanega, C. Hernandez, A. Gianarro, I. Broster, and F. J. Cazorla, "Fitting processor architectures for measurement-based probabilistic timing analysis," *Microprocessors and Microsystems*, vol. 47, pp. 287–302, 2016.

[17] M. Yang, S. Wang, J. Bakita, T. Vu, F. D. Smith, J. H. Anderson, and J.-M. Frahm, "Re-thinking cnn frameworks for time-sensitive autonomous-driving applications: Addressing an industrial challenge," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019, pp. 305–317.

[18] W. Jang, H. Jeong, K. Kang, N. Dutt, and J.-C. Kim, "R-tod: Real-time object detector with minimized end-to-end delay for autonomous driving," in *2020 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2020.

[19] European Union Aviation Safety Agency (EASA), Daedalean, "Concepts of design assurance for neural networks (codann) ipc extract," https://www.easa.europa.eu/downloads/112151/en, 3 2020.

[20] European Union Aviation Safety Agency (EASA), "Artificial intelligence roadmap: A human-centric approach to ai in aviation: Version 1.0," https://www.easa.europa.eu/downloads/109668/en, 2 2020.

[21] European Union Aviation Safety Agency (EASA), Daedalean, "Concepts of design assurance for neural networks (codann) ii," https://www.easa.europa.eu/downloads/128161/en, 5 2021.

[22] European Union Aviation Safety Agency (EASA), "Easa concept paper: First usable guidance for level 1 machine learning applications: A deliverable of the easa ai roadmap," 12 2021. [Online]. Available: https://www.easa.europa.eu/downloads/134357/en

[23] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," *Lecture Notes in Computer Science*, pp. 21–37, 2016.

[24] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.

[25] A. Howard, M. Sandler, B. Chen, W. Wang, L.-C. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam, and Q. Le, "Searching for mobilenetv3," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, 2019.

[26] E. Dubrova, *Fault-tolerant design*. New York, NY and Heidelberg: Springer, 2013.

[27] J. Bin, D. Girbal, Sylvai nand Gracia Pérez, A. G. Grasset, and A. Mérigot, "Studying co-running avionic real-time applications on multi-core COTS architectures," in *ERTS 2014*, 2014.

[28] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*. IEEE, 2007, pp. 239–243.