DTM: Degraded Test Mode for Fault-Aware Probabilistic Timing Analysis

Mladen Slijepcevic*,†, Leonidas Kosmidis*,†, Jaume Abella†, Eduardo Quiñones†, Francisco J. Cazorla†,‡
*Universitat Politècnica de Catalunya
†Barcelona Supercomputing Center
‡Spanish National Research Council (IIIA-CSIC)

***Abstract*—Existing timing analysis techniques to derive Worst-Case Execution Time (WCET) estimates assume that hardware in the target platform (e.g., the CPU) is fault-free. Given the performance requirements increase in current Critical Real-Time Embedded Systems (CRTES), the use of high-performance features and smaller transistors in current and future hardware becomes a must. The use of smaller transistors helps providing more performance while maintaining low energy budgets; however, hardware fault rates increase noticeably, affecting the temporal behaviour of the system in general, and WCET in particular.**

In this paper, we reconcile these two emergent needs of CRTES, namely, tight (and trustworthy) WCET estimates and the use of hardware implemented with smaller transistors. To that end we propose the *Degraded Test Mode (DTM)* that, in combination with fault-tolerant hardware designs and probabilistic timing analysis techniques, (i) enables the computation of tight and trustworthy WCET estimates in the presence of faults, (ii) provides graceful average and worst-case performance degradation due to faults, and (iii) requires modifications neither in WCET analysis tools nor in applications. Our results show that DTM allows accounting for the effect of faults at analysis time with low impact in WCET estimates and negligible hardware modifications.

I. Introduction

There is an increasing need for high guaranteed performance in Critical Real-Time Embedded Systems (CRTES) industry ranging from automotive to space, aerospace, railway and medical industries. To respond to this demand, the use of hardware acceleration features such as caches and smaller technology nodes (i.e. smaller transistors) is required. In particular, the higher degree of integration provided by newer technology nodes enables (i) reducing energy consumption and temperature, and (ii) integrating more hardware functionalities per chip, which in turn enables running more system functionalities per chip, thus reducing overall system size, weight and power consumption costs.

Semiconductor technology evolution (i.e. smaller technology nodes) leads, however, to an increased number of permanent faults manifesting during operation [3]. While test methods can detect most permanent faults during post-silicon testing, many rather small defects not producing actual faults escape the test process (latent faults). Hardware degradation makes those latent defects grow enough to cause faults during operation. Errors induced by permanent faults can be tolerated to certain extent in some markets, but cannot in CRTES where stringent correctness constraints call for means to prevent faults from jeopardising the safety of those systems. While this was not an issue for old technology nodes (e.g., 180nm), it becomes critical for newer nodes (e.g., 65nm) since Failures in Time (FIT)[1] rates may grow quickly after only 10 years of operation [17], which is less than the lifetime required for many CRTES (e.g., aircraft, space missions). Furthermore, smaller technology nodes (45nm and beyond) suffer high FIT rates much earlier. Some existing error detection and correction techniques can cope with both functional and *timing* correctness required in CRTES despite of permanent faults. For instance, triple modular redundancy (TMR) [25] has no effect on the timing behaviour despite of errors, but introduces high overheads since complete hardware blocks (e.g., processing cores) are replicated. Analogously, other approaches based on setting up some degree of redundancy are expensive in the context of CRTES if used extensively to deal with permanent faults [11], [21]. Alternatively, hardware can be reconfigured (e.g., disabling faulty components) when permanent faults arise [2], [31], [40]. However, such reconfiguration provides degraded performance, which is a major concern in CRTES, since CRTES must provide both, functional and *timing* correctness.

Timing analysis techniques have been devised to obtain Worst-Case Execution Time (WCET) estimates of applications running on a particular hardware. The fact that hardware characteristics degrade in an exponential number of different ways due to faults challenges state-of-the-art timing analysis methods. Static Timing Analysis (STA) techniques construct a cycle-accurate model of the system that is fed with a mathematical representation of the code to derive a safe upper-bound of the WCET [39]. STA techniques require detailed knowledge of the underlying hardware. For instance, in the case of cache analysis, the replacement policy (e.g., least-recently used, LRU) and the placement policy (e.g., modulo) used by the hardware need to be known. In general, these models do not take into account any information about the permanent faults that hardware may experience. So, as soon as any of the hardware components (e.g., a cache line) is detected to be faulty and disabled by the fault-tolerance mechanisms in place, the WCET estimates previously derived are no longer a safe upper-bound of the WCET of the application. Similarly, WCET estimates derived with Measurement-Based Timing

[1] 1 FIT corresponds to 1 failure per 10^9 hours of operation.

Analysis (MBTA) techniques or Hybrid ones (i.e. combination of MBTA and STA) [39], remain only valid under the same processor degraded mode on which measurements were taken (typically a fault-free mode).

In summary, it becomes mandatory for timing analysis tools to account for degraded hardware behaviour. However, to the best of our knowledge no timing analysis technique has been developed so that it can safely and tightly provide WCET estimates on top of degraded hardware. Only few works have proposed ad-hoc hardware solutions to keep timing characteristics of hardware despite faults at the expense of some redundancy and extra complexity in cache memories, which experience most hardware faults due to their large area and aggressive transistor scaling to provide further cache space and reduced energy consumption [4], [5].

This paper addresses the challenges of enabling tight and safe WCET estimation on faulty hardware with graceful WCET degradation by proposing the *Degraded Test Mode* (DTM) for cache memories. DTM works in conjunction with probabilistic timing analysis (PTA) [10], [12] that has emerged as an alternative to classic STA and MBTA. PTA allows designers to reason in probabilistic terms about the execution time variations of programs when executed on time-randomised (fault-free) hardware, which is indeed in line with the essence of random appearance of permanent faults.

DTM specifies how hardware must be designed and tested so that faults are tolerated enabling the use of PTA on top of such hardware. In particular, DTM focuses on cache designs, since most faults are expected to affect caches and caches are one of the resources affecting WCET tightness the most. DTM determines the properties cache memories must exhibit and how they must be configured so that they can be analysed with PTA techniques, while obtaining WCET estimates valid in the presence of faults. DTM achieves (i) graceful average performance degradation and (ii) WCET degradation in the presence of faults, while still enabling (iii) trustworthy and tight WCET estimates despite of faults, by (iv) using unmodified PTA tools and (v) introducing no changes in applications so that DTM can be used even for legacy code.

The rest of this paper is organised as follows. Section II provides background on CMOS technology evolution and PTA methods. Section III introduces the fault model and processor architecture used in this study. Section IV describes our approach for fault-tolerant CRTES. Section V evaluates our approach. Section VI presents some related work. Finally, Section VII draws the main conclusions of this work.

II. Background

A. CMOS Technology Evolution

CMOS technology suffers from process variations [9]. Process variations are deviations of device (e.g., transistor, wire) parameters from their nominal values. The relative impact of those variations increases as device geometry shrinks. While this is a challenge for all hardware components, it is particularly critical for cache memories because bit-cells are typically implemented with the smallest transistors allowed (so the relative impact of variations exacerbates). Moreover, variations cannot compensate across devices because bit-cells involve very few transistors (e.g., typical cells are implemented with 6 or 8 transistors only [20]). Deviations due to process variations lead to timing faults (signal transitions take longer than the cycle time), retention faults (bit-cell contents eventually flip from 0 to 1 or vice versa) and read/write faults (bit-cells cannot be properly read or written).

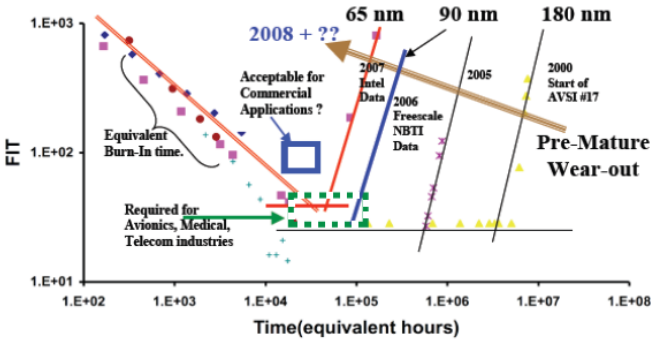

Fig. 1. FIT rates of modern semiconductor technologies used for embedded microprocessors (65nm, 90nm, 130nm and 180nm) over the chip's lifetime. The X and Y axes show equivalent hours of operation and FIT rates respectively in logarithmic scale (source: Jet Propulsion Laboratory, NASA [17]).

Test methods exist to verify that processors are fault-free when delivered. Unfortunately, latent defects are not large enough to cause errors and escape this test process. After some time of operation, degradation makes latent defects to cause actual faults. This effect exacerbates as device geometry shrinks. To illustrate this phenomenon Figure 1 shows the FIT rate for technologies down to 65nm [17]. We can observe that until 10^4 hours of operation the FIT rate decreases. This period is known as *infant mortality* and is elapsed before product deployment through accelerated stress in appropriate equipment. Then, the FIT rate remains low during some time (normal lifetime) until it raises exponentially due to degradation. Rightmost vertical lines in the plot show how this FIT rate ramp up occurs earlier for newer technologies, leading to a too short normal lifetime for some technologies (e.g., few thousands of hours for 65nm). Therefore, the normal lifetime where FIT rates are low enough for safety-critical systems become too short. For instance, while such ramp up would take hundreds of years to occur for 180nm technology nodes, it occurs after 10 years of operation for 65nm. This is already a challenge for avionics and space industry where aircraft and space missions are expected to last several decades (i.e. $8{\cdot}10^4{-}4{\cdot}10^5$ hours), and where the FIT rate is expected to be largely below 100), as shown in the dotted green rectangle in Figure 1. Further, 45nm nodes and beyond are expected to observe a FIT rate ramp up much before 10 years of operation ($8 \cdot 10^4$ hours), thus challenging also other CRTES industries such as automotive and medical ones.

Several techniques exist to perform error detection, correction, diagnosis and reconfiguration, so functional correctness, as required in CRTES, can be achieved. However, modifying hardware behaviour to keep operating in the face of faults

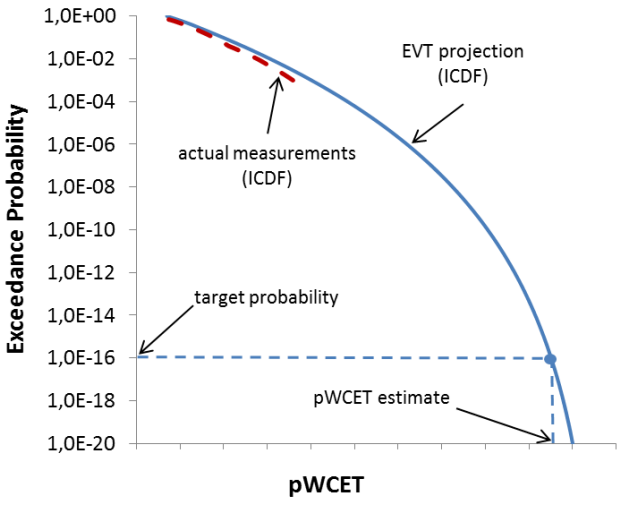

Fig. 2. Example of the ICDF and tail projection.

(e.g., disabling faulty cache lines or decreasing operating frequency) degrades performance. Unfortunately, existing timing analysis methods are unable to provide tight and trustworthy WCET estimates if hardware is expected to become faulty during operation. The fact that tasks are executed correctly but late (finishing after their respective deadlines) can be as catastrophic in CRTES as producing wrong results. Therefore, methods to account for permanent faults in WCET estimates are needed.

B. Probabilistic Timing Analysis

Recently, a new family of timing analysis techniques, probabilistic timing analysis (PTA) [12], [10], has emerged as an alternative to classic approaches. Unlike previous analysis techniques that provide a single WCET figure per task, PTA provides a distribution function that upper-bounds the execution time of the program under analysis, guaranteeing that the execution time of a program only exceeds the corresponding execution time bound with a probability lower than a given target probability (e.g., 10^{-15} per activation). In this way the *probabilistic WCET* (pWCET) is defined as the execution time bound with its corresponding exceedance probability.

Two different families of PTA techniques have been developed so far: static PTA (SPTA) and measurement-based PTA (MBPTA). SPTA [10] derives pWCET estimates by determining the Execution Time Profile (ETP) of each individual operation. The ETP is expressed as a pair of vectors, one describing the different latencies of the operation and the other the probability of occurrence of each latency. In essence, an ETP is the probability distribution function (PDF) of an instruction. Those ETPs can be operated to generate the PDF of the whole program.

MBPTA [12], the technique used in this paper, estimates instead the pWCET based on a collection of observed execution times of the application under analysis. MBPTA is much closer to industrial practice to compute WCET estimates and hence, more appealing to industry. Unlike classic MBTA approaches, that suffer from discontinuous changes in timing due to pathological cache access patterns or other unanticipated timing behaviour, MBPTA imposes several constraints on how the hardware below must behave [22], mainly the randomisation of the timing behaviour of some resources like cache memories. Those constraints make timing analysis insensitive to cache access patterns, which cannot produce timing variations per se. MBPTA, analogously to SPTA, requires the hardware to guarantee that each operation has its own ETP; however, unlike SPTA, which needs to know all ETPs, MBPTA only requires those ETPs to exist. In other words, if the collection of execution times was obtained rolling a dice, SPTA would need to know the number of faces of the dice, their values and their probabilities of occurrence. Conversely, MBPTA would derive pWCET estimates by simply rolling the dice (executing the program and collecting execution times).

MBPTA provides means to determine how many times a program must be run to provide sound results (i.e. pWCET estimates). Then, the collection of execution times feed a tool implementing *Extreme Value Theory* (EVT) [12], [23], which builds a trustworthy and tight upper-bound of the tail of the observed execution time distribution. By doing so, MBPTA provides *pWCET* estimates for arbitrarily low *target probabilities*. Figure 2 shows a hypothetical result of applying EVT to a collection of 1,000 observed execution times. The continuous line represents the inverse cumulative distribution function (ICDF) derived from the observed execution times. The dotted line represents the projection obtained with EVT.

C. pWCET Estimates and Scheduling

pWCET estimates can be used in two different ways for scheduling purposes. The first approach consists of deriving pWCET estimates for an exceedance probability low enough to meet the system safety requirements, which are typically expressed in terms of failures per hour of operation. Note that although low, those probabilities are not null in avionics and automotive safety-related systems, among others, as shown in DO-178B [32] and ISO26262 [19] functional safety standards for avionics software and automotive systems respectively, even for the highest safety levels (DAL-A and ASIL-4 respectively). If pWCET estimates are derived so that software does not violate the corresponding failure per hour threshold[2], then they can be used analogously to WCET estimates derived with non-probabilistic timing analysis, thus having no influence on the particular method used for task scheduling. However, this topic is beyond the scope of this paper. Some further discussion can be found in [10].

An alternative approach consists of using the actual pWCET distributions (a.k.a., the Gumbel functions derived with MBPTA) instead of particular pWCET estimates to perform task scheduling in such a way that probability distributions of the different tasks to be scheduled are considered together. A detailed discussion of how this is achieved can be found in

[2]The failure per hour threshold must be determined according to the safety level of the particular piece of software under analysis as defined in functional safety standards [32], [19].

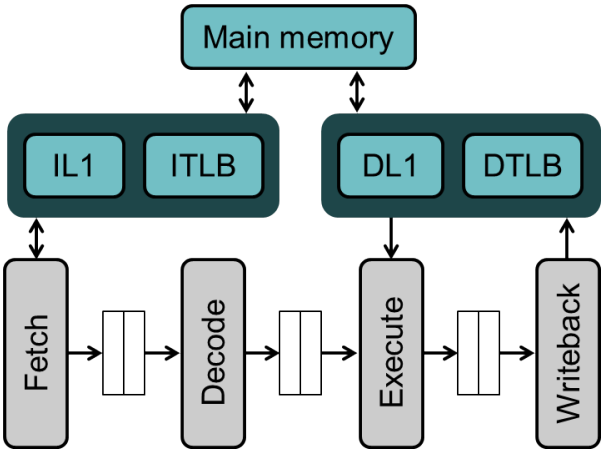

Fig. 3. Processor architecture considered.

[12]. Therefore, pWCET distributions for different tasks are independent[3] and can be used for probabilistic task scheduling [13], [24], [26].

III. Target Processor Architecture and Fault Model

This section introduces the hardware designs upon which we build our approach as well as the fault model used.

A. Hardware Designs for PTA

In [22] we have shown a processor architecture that is analysable with PTA techniques. Our processor architecture consists of a 4-stage (fetch, decode, execute and write-back/commit) in-order processor with first level data (DL1) and instruction (IL1) caches, as well as data (DTLB) and instruction (ITLB) translation lookaside buffers, in line with those used in PTA-related works [10], [12], see Figure 3. DL1, IL1, DTLB and ITLB can be fully-associative, set-associative or direct-mapped as long as they implement random placement and replacement [22]. However, DTM is not constrained to those processor designs. The only constraint is that the processor must be PTA-friendly so that PTA can be used to determine the pWCET for any given exceedance probability. Details about cache configurations, latencies and benchmarks used are provided later in Section V.

B. Caches

Although any component in the processor can be faulty, we only consider faults in cache-like components such as DL1, IL1, DTLB and ITLB since they are expected to concentrate most of the faults as explained in Section II-A. However, the approach proposed in this paper can consider faults in any component as long as those faults do not impact the functional correctness of programs being executed —faults only impact timing once they have been detected and faulty parts reconfigured properly.

[3]Execution time may be dependent across tasks, but pWCET distributions, by construction of the MBPTA method, are derived in such in a way that dependences are upper-bounded.

If cache-like blocks are expected to experience soft errors, they may be parity or Error Detection and Correction (EDC) protected. We consider that those error detection and/or correction features are set up *only* to provide a given degree of fault tolerance in front of soft errors, whose rate can be very high in segments such as avionics and space, where systems operate in high-radiation environments (thousands of meters above the sea level or the space). Permanent faults arising from the use of nanotechnologies cannot be addressed by those means set up to deal with soft errors because that would decrease soft error protection. For instance, if data in cache are Single-error-correction and double-error-detection (SECDED) protected to be able to correct up to one soft error per word, permanent faults can rely on SECDED to detect and correct the error the first time it manifests, but those words must be disabled because a soft error combined with the existing permanent fault could easily lead to a non-correctable double error.

We assume that whenever a faulty bit is detected in cache, the whole cache line is disabled. Such an approach is similar to what we can find in some existing (non-embedded) processors such as the Itanium family, which implements Pellston technology [27] to disable faulty cache lines during operation. A potential implementation of Pellston technology consists of extending each cache line with a *Usable Bit* (UB for short). Such bit must be set for all cache lines but faulty ones. Whenever cache tags are checked on an access, the match signal is ANDed with the UB so that faulty lines cannot report a hit. Note that such AND operation is also performed with the valid bit, which indicates whether the cache line holds valid contents.

C. Permanent Fault Model

In order to model permanent faults caused by process variations and aging in nanotechnologies, we use as input the fault probability per bit ($p(bit)_f$), which is the common practice in fault modelling in caches due to the random nature of those faults [2], [31], [40]. Such $p(bit)_f$ corresponds to the probability of a bit to become permanently faulty during the target lifetime of the chip. We assume faults to be random given that systematic (and hence, non-random) faults can be easily addressed with techniques such as body biasing [36]. Hence, we apply such $p(bit)_f$ homogeneously to all DL1, IL1, DTLB and ITLB bits except the UB bits, which must be hardened to allow faulty lines to be properly disabled. Hardening can be performed, for instance, by using larger transistors or triple modular redundancy. Based on the $p(bit)_f$ and the number of bits per line ($bits_{line}$) we obtain the probability of a line to be fault-free ($p(line)_{ok}$) in each cache-like structure:

$$p(line)_{ok} = (1 - p(bit)_f)^{bits_{line}} \quad (1)$$

Next, we obtain for each cache-like structure the probability of each faulty cache line count: $p(cache)_x$ where x is the number of faulty cache lines and N the total number of cache lines:

Inputs:
Outputs:
(1) C = set of all caches
(2) X = set of faulty entries considered for each cache
(3) for each cache $cache_i \in C$ do
(4) $x_i = 0$ (where $x_i \in X$)
(5) while $(1 - Yield(cache_i)) > TargetFailureRate(chip)$ do
(6) $x_i = x_i + 1$
(7) endwhile
(8) endfor
(9) while $(1 - \prod_{i=1}^{NumCaches} Yield(cache_i)) > TargetFailureRate(chip)$ do
(10) $cache_{max} \in C$ so that $\forall cache_i \in C : Yield(cache_{max}) \leq Yield(cache_i)$
(11) $x_{max} = x_{max} + 1$
(12) endwhile
(13) return X

Fig. 4. Algorithm to obtain the number of faulty entries to consider for each cache.

$$p(cache)_x = (p(line)_{ok})^{N-x} \cdot (1 - p(line)_{ok})^x \cdot \begin{pmatrix} N \\ x \end{pmatrix} \quad (2)$$

The first element in Equation 2 corresponds to the probability of $N - x$ lines not to be faulty, the second element is the probability of having x lines faulty and the third the number of arrangements of x faulty lines in a cache with N lines.

D. Upper-bounding the Number of Faulty Cache Lines

Based on the probabilities of each faulty cache line count, we derive an upper-bound for the number of cache lines that must be assumed to be faulty during the timing analysis to derive safe WCET estimates. For that purpose, we derive the probability of a cache to have up to x faulty lines. We refer to this probability as $Yield(cache)$:

$$Yield(cache) \leq \sum_{i=0}^{x} p(cache)_i \quad (3)$$

where x is the faulty cache line count. Note that Equation 2 gives the probability of an exact number of cache lines (x) to be faulty. Equation 3 accumulates the probability for those counts up to x faulty lines. For instance if $x = 3$, $Yield(cache)$ stands for the probability of having at most 3 faulty lines in cache during the whole lifetime of the chip.

Finally, the yield of the chip, $Yield(chip)$, can be computed based on the yield of its components (cache memories in our case study).

$$Yield(chip) = \prod_{i=1}^{NumCaches} Yield(cache_i) \quad (4)$$

$NumCaches$ stands for the total number of cache memories in the chip. Such yield must be high enough so that the failure rate (obtained as $1 - Yield(chip)$) is below a given target threshold ($TargetFailureRate(chip)$) as shown in Equation 5.

$$TargetFailureRate(chip) \geq 1 - Yield(chip) \quad (5)$$

Note that $TargetFailureRate(chip)$ is very stringent for CRTES (e.g., 10^{-6}, meaning that up to 1 chip every 1,000,000 may have more faults than allowed during its lifetime).

When the chip comprises several cache memories, several combinations of faulty entries per cache may meet the $TargetFailureRate(chip)$. For instance in a processor with DL1, IL1, DTLB and ITLB the maximum number faulty lines that must be considered for each of those cache memories could be $\{3, 3, 2, 1\}$ or $\{4, 3, 1, 1\}$.

The number of faulty cache lines we must assume for each cache component must be determined based on the $TargetFailureRate(chip)$. How to do this simultaneously for all caches is not trivial, so we develop an algorithm to derive the number of faulty lines per cache that must be considered for WCET estimation. We start considering the maximum number of faulty lines per cache memory such that each particular cache in isolation does not exceed the target failure rate for the whole chip. If the combined yield of the different caches exceeds the target failure rate, then we need a way to increase the number of faulty lines considered in the different caches so that $Yield(chip)$ is increased. For that purpose we propose a simple iterative algorithm to deal with those scenarios. The algorithm is presented in Figure 4. First, the number of faulty cache lines to be considered for each cache in isolation is computed (lines 3-8). Then, the yield of each cache in isolation is combined. If the combined failure rate ($1 - Yield(chip)$) exceeds the threshold (line 9), an extra faulty entry is assumed for the cache with the lowest yield (lines 10-11). This process repeats until their combined failure rate is below the target threshold (lines 9-12).

IV. Making Timing Analysis Aware of Hardware Faults

A. Deterministic Hardware

STA techniques require detailed knowledge of the underlying hardware. For instance, in the case of cache analysis, STA must be aware of the replacement policy (e.g., LRU) and

the placement policy (e.g., modulo) used by the hardware. From the application executable, STA needs the address of each cache access to be able to determine whether they will hit or miss in cache.

Let us assume that, based on the model presented in Section III-C, we know that the cache deployed in a given platform is expected to have up to f faults during its lifetime. In order to make STA tools aware of faults at hardware level we should consider all combinations of f faults over N total lines in cache, leading to a total of $\frac{N!}{f!\cdot(N-f)!}$ different degraded cache modes[4]. For instance, if up to 2 faults are expected ($f = 2$) in a small cache with 64 cache lines ($N = 64$), STA should consider 2,016 different scenarios. Under each degraded mode, cache analysis should be carried out deriving a WCET bound. The highest of those WCET bounds should be taken as the final WCET bound. MBTA techniques would suffer similar problems, since the particular lines in which faults appear affect the analysis.

Overall, conventional caches deploying deterministic placement and replacement policies such as modulo and LRU make WCET estimates provided by STA and MBTA depend on the particular location in which faults occur. In the presence of f faults in a cache with N total cache lines, all possible $\frac{N!}{f!\cdot(N-f)!}$ degraded modes must be considered by the timing analysis technique.

B. Probabilistic (Time-Randomised) Hardware

Randomised caches (i.e. caches deploying random placement and random replacement) [22] break the dependence between the location in memory of a piece of data and its assigned set in cache.

The main property of PTA-friendly (a.k.a. randomised) cache designs is that they are insensitive to the particular location of faults. This drastically simplifies capturing the effect of faulty lines with PTA techniques. In particular, the only information to take into account by PTA techniques to handle faulty cache lines is *the number of faulty lines in each cache-like structure, not their location in cache (i.e. the particular way and set in which the fault line appears)*. We illustrate this phenomenon in the following subsections, starting with a fully-associative random-replacement cache and then extending it to more generic set-associative caches.

1) Fully-associative caches: Random replacement is performed by generating, on a miss, a random number of $\log_2 N$ bits where N is the number of cache lines. On an eviction, the random number obtained can correspond to the cache line number of a faulty cache line. If this is the case, a new random number is generated until the identifier of a fault-free cache line is generated. Note that the expected number of attempts (ATT) required for a cache with N total cache lines and f faulty cache lines (where $f < N$) to pick a fault-free cache line is as follows [14]:

$$ATT = \sum_{i=0}^{\infty} \left(\frac{N-f}{N}\right) \cdot \left(\frac{f}{N}\right)^i \cdot (i+1) = \frac{N}{N-f} \quad (6)$$

In the equation, $\left(\frac{N-f}{N}\right)$ is the probability of picking a fault-free line, and $\left(\frac{f}{N}\right)^i$ the probability of picking a faulty line. Hence, the equation adds each number of attempts $(i+1)$ weighted by the probability of i failed attempts followed by 1 successful attempt. This can be expressed simply as $\frac{N}{N-f}$.

For instance, in a cache with $N = 64$ cache lines and $f = 2$ faulty lines we would need around 1.03 attempts on average. Those extra attempts are unlikely to impact the execution time because they can occur in parallel with the memory access. In our example where 2 out of 64 cache lines are faulty, 1 extra attempt is needed with 0.03 probability, 2 extra attempts with 0.001, 3 extra attempts with 0.00003, and i extra attempts with $\left(\frac{N-f}{N}\right) \cdot \left(\frac{f}{N}\right)^i$ probability.

Note that those cache designs considered for DTM (caches implementing random placement and replacement with re-try support for evictions of faulty lines), differently to existing approaches [4], [5], neither need any cache line redundancy nor additional structures. Moreover, as shown later, the proposed cache designs can be considered in the context of PTA because whether a faulty cache line is chosen for replacement depends solely on a random event.

2) Set-associative caches: If the cache under consideration is set-associative, Equation 6 applies at the level of cache set, with N being the associativity of the cache (a.k.a. the number of cache lines per set). A further consideration must be done for set-associative caches, especially if their degree of associativity is low. If the number of potential faulty cache lines is equal or larger than the associativity of such a cache, then exists a non-null probability of having a cache set without any fault-free cache line. If this is the case, accesses to that cache set will be processed as misses and, obviously, not cached. However, given that the number of faulty lines to consider is pretty low, as shown later, and the probability of concentrating many faulty cache lines in a single set is also very low, this scenario is extremely unlikely — if at all possible.

As stated before, randomised caches make PTA and DTM insensitive to the location of faults in a particular cache-like structure. Instead, *what really matters for PTA and DTM is the number of faulty lines in each cache-like structure, not their location*, so the fault model in Section III captures exactly the relevant information for PTA and DTM.

C. DTM: Applying MBPTA on Top of Faulty Hardware

Once determined the number of faulty entries that must be considered in each cache structure, see Section III-D, MBPTA must be properly used to determine the pWCET of the program in such a potentially faulty chip.

We collect execution times of the application as dictated by MBPTA [12], but on top of the hardware with maximum

[4]The particular location of the faulty line matters given that its particular cache set will have one line less than the other sets, and the replacement policy will select cache lines for replacement regardless of their faultiness, thus leading to different scenarios depending on the particular location of the faulty cache line in the set.

degradation. In other words, execution times must be collected on top of a processor with as many cache entries disabled in each cache structure as determined by the process in Section III-D. For that purpose we propose enabling a *Degraded Test Mode* (DTM for short) that allows disabling a number of cache entries in each cache structure.

The DTM affects some parameters of the processor operation similar to those set up in the BIOS or to those that can be configured dynamically in many processors such as the operating frequency. Therefore, DTM can be configured through the BIOS at boot time or through special purpose registers modified during operation. If the latter approach is used, whenever a cache line is deactivated its contents must be written back to the corresponding level of the memory hierarchy if they are dirty.

Given that PTA (and hence MBPTA) is insensitive to the location of faults, simply configuring the number of disabled entries per structure suffices and any implementation of such disabling process will produce equivalent results from a PTA perspective. For instance, cache entries disabled in a fully-associative cache can be either consecutive or randomly located. In a set-associative, whether several faults occur in the same cache set may impact execution time. Thus, it is still irrelevant in which particular cache set faults occur, but not whether they occur in the same or different cache sets. In particular, in this work we assume that disabled cache lines are randomly chosen. The particular lines disabled are changed randomly before each execution of the program. This change can be performed with a specific instruction set up in purpose.

Once the appropriate number of entries is disabled in each cache structure, MBPTA can be used as in fault-free processors running the program under analysis as needed. *By disabling a number of entries in each of the cache memories, DTM allows collecting execution times in a fault-free processor as if it was faulty so that pWCET estimates are safe regardless of whether hardware operation has been degraded due to faults.*

Although DTM is only described for faulty caches (our case study in this paper), the same rationale can be used to deal with faults in other components. For instance, many permanent faults do not make circuits to produce wrong results but to produce correct results too late (after the end of the cycle). Increasing the cycle time while keeping the operating voltage unmodified suffices for those circuits to provide the correct result before the end of the cycle. Thus, analogously to the cache entry disabling approach, DTM could also increase the cycle time as needed to cover other fault types. Nevertheless, how to deal with faults in components others than cache memories is left as future work.

V. Evaluation

This section describes the evaluation framework and provides results proving the suitability of the proposed platform to be used in the context of faulty processors.

A. Evaluation Framework

Execution times have been collected with a cycle-accurate execution-driven simulator based on the SoCLib simulation framework [35], with PowerPC binaries [38]. Two different configurations have been considered:

1) Both DL1 and IL1 caches are 2KB fully-associative random replacement 32B/line caches. Both the DTLB and ITLB are 16-entry fully-associative random replacement TLBs for page sizes of 1KB.
2) Both DL1 and IL1 caches are 2KB 4-way set-associative random placement and replacement 32B/line caches. Both the DTLB and ITLB are 16-entry 4-way set-associative random placement and replacement TLBs for page sizes of 1KB.

Cache sizes are deliberately small so that disabling some cache lines due to faults has some significant effect in performance given that benchmarks used have particularly small working sets. Hit latencies of 2 cycles and memory latencies of 100 cycles have been considered for all caches.

We use the EEMBC Autobench benchmark suite [29] in this study. EEMBC Autobench is a well-known benchmark suite for some CRTES in the automotive domain.

TABLE I
Maximum number of faulty lines expected for a target yield of 1 faulty part per million (ppm)

$p(bit)_f$	DL1	IL1	DTLB	ITLB
10^{-8}	1	1	1	1
10^{-7}	2	2	1	1
10^{-6}	3	3	1	1
10^{-5}	4	4	2	2
10^{-4}	10	10	3	3

B. Worst-Case Execution Time

Based on the fault model described in Section III-C we have obtained how many DL1, IL1, DTLB and ITLB entries can be faulty to guarantee that at most one out of one million chips has more failures than budgeted during its lifetime or, in other words, the failure rate during the lifetime of the chip is below 10^{-6}. Results are shown in Table I. For instance, for a faulty bit probability ($p(bit)_f$) of 10^{-5} up to 4 DL1 and 4 IL1 cache lines could be faulty (out of the 64 cache lines in each cache) and up to 2 DTLB and 2 ITLB entries could be faulty (out of the 16 entries in each TLB).

Tables II and IV report pWCET relative results for fully-associative and set-associative caches respectively. Absolute values in terms of execution cycles are shown in Tables III and V. The minimum number of runs per benchmark has been computed with the method reported in [12] and never exceeded 1,000 runs. *Note that this is the same number of runs needed for a fault-free system, thus keeping pWCET estimation cost low*. Similarly, independence and identical distribution tests described in [12] have been run and all data passed those tests successfully, as needed by MBPTA.

The 2^{nd} column in tables II and IV shows the pWCET of a fault-free chip for an exceedance threshold of 10^{-15}

TABLE II
pWCET NORMALISED RESULTS FOR FULLY-ASSOCIATIVE CACHES

	pWCET fault-free	**pWCET** $p(bit)_f$ 10^{-8}	**pWCET** $p(bit)_f$ 10^{-7}	**pWCET** $p(bit)_f$ 10^{-6}	**pWCET** $p(bit)_f$ 10^{-5}	**pWCET** $p(bit)_f$ 10^{-4}
	vs avg	**vs pWCET fault-free**				
a2time	1,086	1,060	1,120	1,185	1,250	1,614
aifirf	1,035	1,012	1,021	1,014	1,031	1,074
cacheb	1,093	1,027	1,002	1,019	1,010	1,039
canrdr	1,035	0,990	0,990	0,995	1,002	1,004
puwmod	1,014	1,002	1,004	0,999	1,000	1,005
rspeed	1,035	1,017	1,005	1,004	0,998	1,009
tblook	1,057	1,048	1,063	1,119	1,151	1,369
ttsprk	1,034	1,015	1,011	1,004	1,003	1,013
AVG	1,049	1,022	1,027	1,042	1,056	1,141

TABLE III
pWCET ABSOLUTE RESULTS FOR FULLY-ASSOCIATIVE CACHES (SHOWN IN CYCLES)

	pWCET fault-free	**pWCET** $p(bit)_f$ 10^{-8}	**pWCET** $p(bit)_f$ 10^{-7}	**pWCET** $p(bit)_f$ 10^{-6}	**pWCET** $p(bit)_f$ 10^{-5}	**pWCET** $p(bit)_f$ 10^{-4}
a2time	1464176	1552730	1640469	1735297	1829608	2362593
aifirf	3271249	3310865	3340027	3316892	3371230	3512392
cacheb	328332	337346	329058	334661	331500	341158
canrdr	962489	952498	953340	957943	964564	966324
puwmod	1456856	1460308	1462586	1455496	1456380	1463911
rspeed	320614	326125	322151	321850	320017	323495
tblook	2769816	2902933	2943016	3099158	3188689	3791794
ttsprk	673568	683652	680991	676458	675708	682399

TABLE IV
pWCET NORMALISED RESULTS FOR SET-ASSOCIATIVE CACHES

	pWCET fault-free	**pWCET** $p(bit)_f$ 10^{-8}	**pWCET** $p(bit)_f$ 10^{-7}	**pWCET** $p(bit)_f$ 10^{-6}	**pWCET** $p(bit)_f$ 10^{-5}	**pWCET** $p(bit)_f$ 10^{-4}
	vs avg	**vs pWCET fault-free**				
a2time	4,709	1,157	1,190	1,301	1,693	2,656
aifirf	4,356	1,046	1,075	1,147	1,265	1,335
cacheb	2,351	1,017	1,051	1,143	1,148	1,179
canrdr	1,160	1,007	1,033	1,060	1,065	1,101
puwmod	1,037	0,990	0,997	1,018	1,028	1,031
rspeed	1,049	1,011	1,059	1,070	1,136	1,143
tblook	2,536	1,105	1,135	1,139	1,328	1,358
ttsprk	1,159	1,029	1,032	1,040	1,047	1,069
AVG	2,295	1,045	1,071	1,115	1,214	1,359

TABLE V
pWCET ABSOLUTE RESULTS FOR SET-ASSOCIATIVE CACHES (SHOWN IN CYCLES)

	pWCET fault-free	**pWCET** $p(bit)_f$ 10^{-8}	**pWCET** $p(bit)_f$ 10^{-7}	**pWCET** $p(bit)_f$ 10^{-6}	**pWCET** $p(bit)_f$ 10^{-5}	**pWCET** $p(bit)_f$ 10^{-4}
a2time	1525818	1764867	1815456	1985289	2583766	4053214
aifirf	3375671	3529318	3630212	3872052	4269072	4508195
cacheb	335526	341327	352644	383604	385350	395453
canrdr	1329893	1338785	1373266	1409029	1415855	1463565
puwmod	1564112	1548370	1559503	1592263	1608338	1611912
rspeed	324743	328413	344026	347618	368773	371303
tblook	5186784	5730594	5886359	5908220	6885710	7041307
ttsprk	795970	819253	821154	827911	833493	850845

per run normalised to the average execution time for a fault-free chip[5]. The exceedance threshold indicates how often a run of the program will exceed the pWCET value (in our case once every 10^{15} runs). Our selection of the exceedance threshold, i.e. the probability that an instance of a task misses its deadline, is based on the observation that for the aerospace commercial industry at the highest integrity level DAL-A[6] the maximum allowed failure rate in a piece of software is 10^{-9} per hour of operation [1]. In current implementations, the highest frequency at which a task can be released is 20 milliseconds (up to 1.8×10^5 times per hour) [1]. Hence, the highest allowed failure rate per task activation is 5.6×10^{-15}, which is above our exceedance probability.

We can observe in the 2^{nd} column of Table II that the pWCET with fault-free caches degrades only between 1% and 10% across benchmarks (5% on average) w.r.t. the average performance for fully-associative caches in line with results in [12]. Conversely, the pWCET for a set-associative cache is significantly larger than the average execution time as described in [22] (2^{nd} column of Table IV). The reason behind this behaviour is the fact that fully-associative caches perform a random eviction on every miss. Instead, set-associative caches randomly map each address into a cache set, and such mapping holds during the whole run. Therefore, if some addresses collide in a particular cache set, this behaviour will hold during the whole execution, so its impact in execution time will be larger. Such larger variation requires a higher pWCET upper-bound than for fully-associative caches. This can be easily explained with an example. Let assume we flip 3 coins. If we flip each one individually (random replacement), then we have 8 different sequences of events. Instead, if we attach the 3 coins with glue (e.g., faces in one side and tails in the other side) and then we flip them (random placement), only 2 different events can occur. Thus, although both approaches are equally random, the potential scenarios that they can generate are different (random placement can effectively generate only a subset of those scenarios produced by random replacement) and so their probabilities.

Columns from 3^{rd} to 7^{th} in Tables II and IV show the pWCET degradation w.r.t. the pWCET of a fault-free chip for different $p(bit)_f$ values. As shown most of the benchmarks observe negligible pWCET degradation because they do not fully exploit all cache space[7] and suffer no degradation due to the deactivation of few cache entries. This is particularly true for fully-associative caches (Table II). In fact *canrdr*

[5]Note that the exceedance probability (10^{-15} per run in our case) and the faulty bit rate ($p(bit)_f$) stand for different concepts. $p(bit)_f$ determines the number of faulty cache lines that must be considered for each cache memory. Then, those (degraded) cache memories are used to obtain execution times, which are the input of MBPTA. Finally, given the pWCET distribution provided by MBPTA, we pick as pWCET estimate the pWCET value corresponding to the particular exceedance probability required.

[6]DAL stands for Design Assurance Level and ranges between A (catastrophic effects) and E (no effects).

[7]Some benchmarks like *aifirf* do not fit in cache, so they reuse few cache lines. In this case, most of the cache lines become useless, as it is the case of those benchmarks needing less cache space than available.

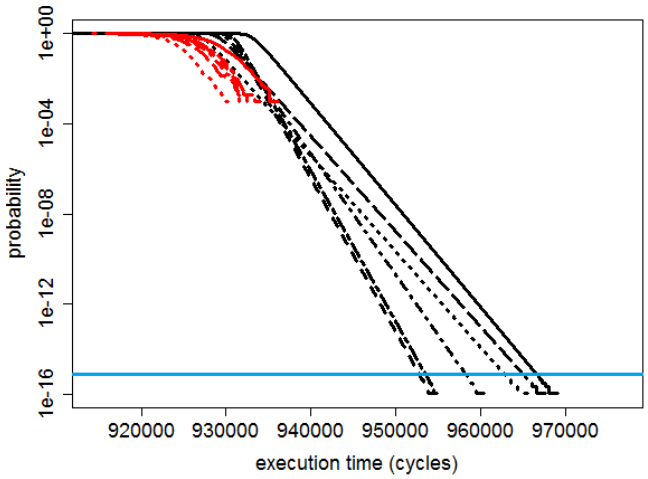

Fig. 5. Inverse cumulative distribution function (ICDF) of pWCET curves and actual observations for *canrdr* benchmark under a fully-associative random-replacement cache. At the 10^{15} probability cutoff point, from left to right, cache configurations cross in the following order: $p(bit)_f = 10^{-8}$, $p(bit)_f = 10^{-7}$, $p(bit)_f = 10^{-6}$, *fault-free*, $p(bit)_f = 10^{-5}$, $p(bit)_f = 10^{-4}$.

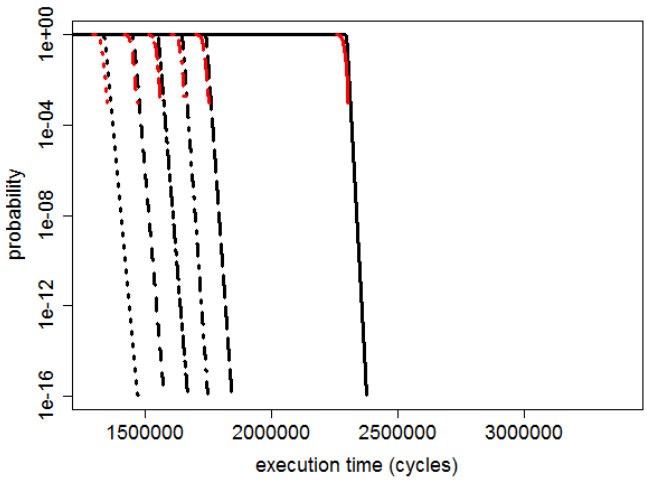

Fig. 6. Inverse cumulative distribution function (ICDF) of pWCET curves and actual observations for *a2time* benchmark under a fully-associative random-replacement cache.

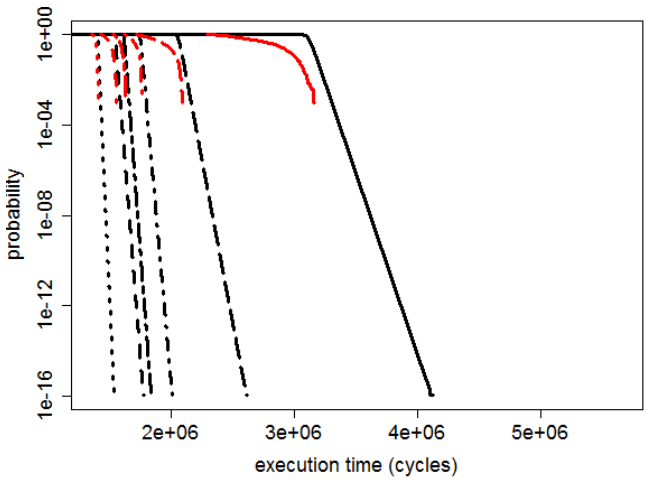

Fig. 7. Inverse cumulative distribution function (ICDF) of pWCET curves and actual observations for *a2time* benchmark under a set-associative random-placement random-replacement cache.

observes some negligible pWCET reductions in some cases. This occurs because of the particular 1,000 execution times in the sample, but results are still safe and tight. We show the ICDF for the different $p(bit)_f$ pWCET curves of *canrdr* together with actual execution times collected for a fully-associative random-replacement cache in Figure 5. At 10^{-15} exceedance probability, the *fault-free* case, which one would expect to provide the lowest pWCET value, provides in fact the fourth lowest value. However, as already discussed in [12], this can happen because MBPTA builds upon a finite random sample. Therefore, although the method provides a Gumbel distribution upper-bounding the tail of the actual execution time distribution, the tightness of the particular distribution may vary depending on the actual observations in the sample. In the particular case of *canrdr*, cache size has negligible impact in performance and, therefore, having fewer cache lines has an effect on performance largely below that of the actual random events occurring during the execution (i.e. random replacement of cache lines). In the particular case of the *fault-free* cache, this leads to a larger σ value for the Gumbel distribution, which increases the overestimation as the exceedance probability decreases. Note, however, that such overestimation is around only 1% higher than the actual overestimation[8] for $p(bit)_f = 10^{-8}$, $p(bit)_f = 10^{-7}$ and $p(bit)_f = 10^{-6}$ cases.

pWCET estimates increase for cache-sensitive benchmarks (*a2time* and *tblook*) in the fully-associative configuration as shown in Table II. A maximum pWCET increase of 61% is needed for *a2time* despite the high $p(bit)_f$ values considered due to the performance robustness of PTA-friendly cache designs. pWCET estimates for set-associative caches (see Table IV) grow faster with faulty bit rates due to the increased variability caused by those cache lines being disabled. This effect is particularly noticeable for those benchmarks using efficiently the cache space available such as *a2time*, *aifirf* and *tblook*. However, even for high $p(bit)_f$ values (e.g., $p(bit)_f = 10^{-5}$), pWCET degrades only 21% on average with respect to the fault-free scenario.

For the sake of illustration, we show the different ICDF curves for *a2time* for both cache configurations in Figure 6 (fully-associative cache) and Figure 7 (set-associative cache). In both cases curves appear from left to right, starting with the lowest $p(bit)_f$ value (0 for the *fault-free* case) and ending with the highest $p(bit)_f$ value ($p(bit)_f = 10^{-4}$). As already shown in [12], pWCET distributions upper-bound actual execution times in the probability range of those execution times (i.e. down to 10^{-3} exceedance probability per run in

[8]Note that the actual overestimation cannot be determined because computing the exact execution time distribution is unaffordable in general. In [12] authors prove for a particularly simple scenario that such overestimation is rather low.

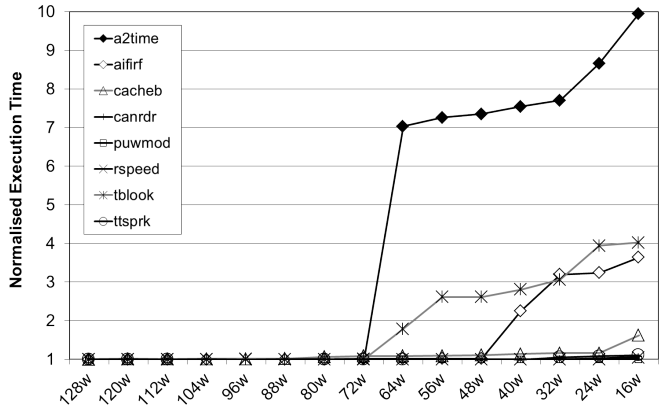

Fig. 8. Normalised execution time for fully-associative LRU-replacement caches with respect to a 128-line cache.

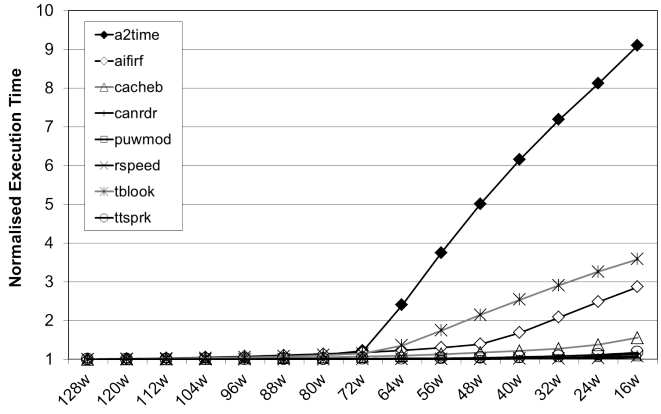

Fig. 9. Normalised execution time for fully-associative random-replacement caches with respect to a 128-line cache.

our case). pWCET curves upper-bound real distributions for any exceedance probability. Their tightness depends on the actual variability existing in the execution times collected. For instance, variability is low in the case of fully-associative caches because each replacement occurs independently as reported in [22]. This leads to very tight pWCET estimates. However, if variability is higher (e.g., the case of a random-placement cache when $p(bit)_f = 10^{-4}$), MBPTA ends up deriving a Gumbel distribution accounting for such variability in the form of a higher σ value, which translates into a gentler slope and so, potentially less tight pWCET estimates.

In summary, PTA together with PTA-friendly cache designs are particularly suitable to obtain time-robust and time-analysable fault-tolerant hardware designs as needed for DTM. *DTM enables for the first time the computation of WCET estimates on top of faulty hardware at very low cost.*

C. Average Performance

This section analyses the average performance for fully-associative caches. While this could be done also for set-associative caches, fully-associative ones allow varying cache size at fine granularity (e.g., removing one cache line), whereas set-associative caches could only be studied at a coarser granularity (e.g., removing one cache way or half of the cache sets). Nevertheless, conclusions for fully-associative caches can be easily extrapolated to set-associative ones.

As stated before, caches implementing LRU replacement are not suitable for PTA and, instead, random-replacement (RR for short) ones must be used. In this section we show that RR caches provide performance comparable to LRU ones and, on top of that, provide much better average performance degradation when cache size is decreased. Such a feature is particularly interesting for WCET analysis on top of faulty caches because disabling few cache lines due to faults has relatively low impact in performance.

Figures 8 and 9 show the degradation of the execution time of several EEMBC benchmarks when the number of cache lines for DL1 and IL1 caches decreases for LRU and RR caches respectively. Cache size is decreased by 8 lines in each step from 128 (4KB) down to 16 cache lines (512B). Although such granularity is coarser than the number of faulty cache lines one may expect in those caches, it serves to illustrate how LRU caches experience abrupt performance degradation when the working set does not fit in cache, whereas such degradation is much smoother for RR caches. Although not shown, we have also observed that decreasing cache size by 1 line in each step still provides smooth performance degradation for RR caches and abrupt changes for LRU ones. For instance, the epitome example is *a2time*, whose execution time grows by 7X when reducing cache size from 72 to 64 caches lines. RR caches suffer the same relative performance degradation when reducing cache size from 72 to 32 cache lines, thus smoothing the effect of some cache lines being disabled. At a lower scale *aifirf* and *tblook* show similar effects. By smoothing the effect of disabling few cache lines, RR caches allow DTM to provide pWCET estimates close to those of fault-free systems despite of faults (see Table II).

Note that while results for LRU caches are deterministic, this is not the case for RR caches, so we report the average execution time across 1,000 runs for each benchmark and configuration for RR caches. In all cases the standard deviation of the execution times for RR is below 0.5%.

The increase in terms of average execution time is very similar to the increase in pWCET. For instance, average execution time increases by 28% and 14% for *a2time* and *tblook* respectively when decreasing the number of available cache lines from 64 down to 60. For $p(bit)_f = 10^{-5}$, where 4 cache lines are disabled from both DL1 and IL1, *a2time* and *tblook* respective pWCET estimates grow by 25% and 15%.

Figure 10 compares RR caches versus LRU ones showing that PTA-analysability and graceful performance degradation do not come at the expense of a significant performance drop. The figure depicts for each benchmark the slowdown/speedup for each cache size (from 128 to 16 cache lines from left to right for each benchmark). Values higher than 1 indicate RR slowdowns (above the red line) and speedups otherwise. For

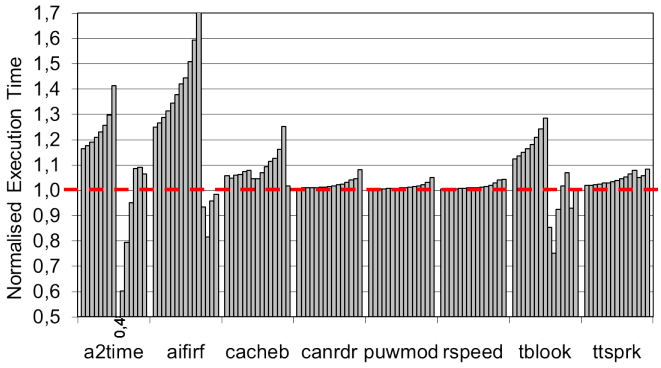

Fig. 10. Normalised execution time of random-replacement caches with respect to LRU ones.

instance, *a2time* slowdown of RR caches versus LRU ones grows from 16% to 41% when moving from 128 to 72 cache lines; however, when moving to 64 cache lines there is a 60% speedup that decreases as we keep decreasing cache size. Overall, RR execution time across benchmarks and cache sizes is only 7% higher than that of LRU caches.

To summarise, the proposed platform allows delivering probabilistically time-analysable fault-tolerant CRTES whose average execution time is comparable to that of conventional systems.

VI. Related Work

Timing analysis of systems equipped with cache memories is a serious challenge even for fault-free systems. The impact of caches on WCET has been extensively studied by the research community [15][28][30], but those techniques show very limited scalability and prevent the adoption of increasingly complex hardware and software.

Some approaches exist (or are under elaboration) to detect errors and recover while keeping time predictability [8], [18]. Those approaches consider the impact in task scheduling of error detection and recovery, and schedule those activities smartly to prevent deadline misses. Furthermore, how to deal with errors at different layers and how to minimise the cost of error detection and recovery is also considered. Unfortunately, the effect in timing of permanent faults is not addressed, so if some hardware resources need to be disabled or reconfigured due to permanent faults, WCET estimates are no longer valid.

There are several hardware solutions to keep time predictability despite of permanent faults. Some of them are based on setting up spares to physically replace faulty entries [21]. Those solutions are typically expensive due to the redundant resources and the costly fuses required to physically reprogram circuits. Another approach consists of using error detection and correction (EDC) codes [11]. This family of solutions is also costly because permanent faults require using EDC logic on each access (energy overhead) and delaying the delivery of data until EDC logic generates a safe output value. A different approach consists of adding some assist structures to perform a *soft* replacement of faulty entries [4], [5]. This has been proposed for cache memories where victim caches, eviction buffers or similar cache-assist structures are conveniently modified to replace faulty entries while keeping time predictability as needed for WCET analysis, but some extra redundancy and design complexity is introduced.

Alternatively to conventional timing analysis, probabilistic timing analysis (PTA) shows promising results [10], [12] for complex platforms running complex applications [37]. Those approaches rely on platforms providing some properties so that execution time variations caused by the hardware itself depend solely on random events. This is achieved, for instance, by using random-replacement caches.

Some existing embedded processors already implement random-replacement policies [6], [7]. Randomised caches in high-performance processors were first proposed in [33] to remove pathological cases produced by the systematic cache misses generated in bad strides. To do so, authors used a pseudo-random hash function to randomise addresses into cache sets, developing an analytic approach to determine cache performance. Other cache designs have attempted to remove cache conflicts by changing the placement function [16] and/or combining several placement functions for different cache ways [34]. However, those caches still produce deterministic conflicts across addresses. Recently, direct-mapped and set-associative cache designs implementing random placement have been proposed [22]. Those designs have been successfully proven to fit PTA needs.

Unfortunately, the advent of nanotechnologies poses serious challenges to perform timing analysis while keeping analysis costs low and WCET estimates for applications low, safe and tight. To the best of our knowledge this paper proposes the first methodology, DTM, allowing CRTES to provide those low, safe and tight WCET estimates needed in the CRTES arena despite of faults.

VII. Conclusions

The increasing performance demand for critical real-time embedded systems (CRTES) pushes for more complex hardware implemented with unreliable nanotechnologies. Providing low, safe and tight worst-case execution time (WCET) bounds for complex software on top of complex hardware has been recently addressed by means of probabilistic timing analysis (PTA). Unfortunately, PTA has been devised only for fault-free platforms and does not consider how to deal with the effect of frequent permanent faults in nanotechnologies.

This paper addresses functional and timing correctness in a holistic way by proposing the *Degraded Test Mode (DTM)*, a method to use probabilistically-analysable fault-tolerant hardware designs in combination with PTA techniques. The proposed platform delivers (i) probabilistically time-analysable fault-tolerant CRTES (ii) whose WCET is low, safe and tight, and (iii) whose average execution time is comparable to that of conventional systems. Therefore, CRTES can be implemented on top of unreliable hardware despite of faults, thus increasing CRTES performance and reducing their costs.

ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under the PROARTIS Project (www.proartis-project.eu), grant agreement no 249100. This work has also been partially supported by the Spanish Ministry of Science and Innovation under grant TIN2012-34557 and the HiPEAC Network of Excellence. Leonidas Kosmidis is funded by the Spanish Ministry of Education under the FPU grant AP2010-4208. Eduardo Quiñones is partially funded by the Spanish Ministry of Science and Innovation under the Juan de la Cierva grant JCI2009-05455.

REFERENCES

[1] Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment. *ARP4761*, 2001.

[2] J. Abella, J. Carretero, P. Chaparro, X. Vera, and A. Gonzalez. Low vccmin fault-tolerant cache with highly predictable performance. In *MICRO*, 2009.

[3] J. Abella, F.J. Cazorla, E. Quinones, D. Gizopoulos, A. Grasset, S. Yehia, P. Bonnot, R. Mariani, and G. Bernat. Towards improved survivability in safety-critical systems. In *International On-Line Testing Symposium (IOLTS)*, 2011.

[4] J. Abella, E. Quinones, F.J. Cazorla, Y. Sazeides, and M.Valero. RVC-Based time-predictable faulty caches for safety-critical systems. In *IOLTS*, 2011.

[5] J. Abella, E. Quinones, F.J. Cazorla, Y. Sazeides, and M. Valero. RVC: A mechanism for time-analyzable real-time processors with faulty caches. In *HiPEAC Conference*, 2011.

[6] Aeroflex Gaisler. *Quad Core LEON4 SPARC V8 Processor - LEON4-NGMP-DRAFT - Data Sheet and Users Manual*, 2011.

[7] ARM. *Cortex-R4 and Cortex-R4F Technical Reference Manual*, 2006.

[8] Philip Axer, Maurice Sebastian, and Rolf Ernst. Reliability analysis for MPSoCs with mixed-critical, hard real-time constraints. In *CODES+ISSS*, 2011.

[9] K. Bowman, S. Duvall, and J. Meindl. Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration. *IEEE Journal of Solid-State Circuits*, 2, 2002.

[10] Francisco J. Cazorla, Eduardo Quinones, Tullio Vardanega, Liliana Cucu, Benoit Triquet, Guillem Bernat, Emery Berger, Jaume Abella, Franck Wartel, Michael Houston, Luca Santinelli, Leonidas Kosmidis, Code Lo, and Dorin Maxim. Proartis: Probabilistically analysable real-time systems. *ACM Transactions on Embedded Computing Systems*, (to appear (http://hal.inria.fr/hal-00663329)), 2012.

[11] C.L. Chen and M.Y. Hsiao. Error-correcting codes for semiconductor memory applications: A state of the art review. *IBM Journal of Research and Development*, 28(2):124–134, 1984.

[12] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzeti, E. Quinones, and F.J. Cazorla. Measurement-based probabilistic timing analysis for multi-path programs. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2012.

[13] J.L Díaz, D.F. Garcia, K. Kim, C.G. Lee, L.L. Bello, López J.M., and O. Mirabella. Stochastic analysis of periodic real-time systems. In *the 23rd IEEE Real-Time Systems Symposium (RTSS02)*, 2002.

[14] W. Feller. *An introduction to Probability Theory and Its Applications*. 1996.

[15] Christian Ferdinand, Reinhold Heckmann, Marc Langenbach, Florian Martin, Michael Schmidt, Henrik Theiling, Stephan Thesing, and Reinhard Wilhelm. Reliable and precise wcet determination for a real-life processor. *EMSOFT*, 2001.

[16] A. González, M. Valero, N. Topham, and J.M. Parcerisa. Eliminating cache conflict misses through XOR-based placement functions. In *International Conference on Supercomputing (ICS)*, 1997.

[17] S. Guertin and M. White. CMOS reliability challenges the future of commercial digital electronics and NASA. In *NEPP Electronic Technology Workshop*, 2010.

[18] Jörg Henkel et al. Design and architectures for dependable embedded systems. In *CODES+ISSS*, 2011.

[19] International Organization for Standardization. *ISO/DIS 26262. Road Vehicles – Functional Safety*, 2009.

[20] Sanjeev K. Jain and Pankaj Agarwal. A low leakage and snm free sram cell design in deep sub micron cmos technology. In *VLSID*, 2006.

[21] I. Koren and Z. Koren. Defect tolerance in vlsi circuits: techniques and yield analysis. *Proceedings of the IEEE*, 86(9):1819–1838, 1998.

[22] Leonidas Kosmidis, Jaume Abella, Eduardo Quinones, and Francisco J. Cazorla. A cache design for probabilistically analysable real-time systems. In *DATE*, 2013.

[23] Samuel Kotz and Saralees Nadarajah. *Extreme value distributions: theory and applications*. World Scientific, 2000.

[24] J.M. Lopez, J. L. Diaz, J. E., and D. Garcia. Stochastic analysis of real-time systems under preemptive priority-driven scheduling. *Real-time Systems*, 40(2):180–207, 2008.

[25] R.E. Lyons and W. Vanderkulk. The use of triple modular redundancy to improve computer reliability. *IBM Journal of Research and Development*, 6(2):200–209, 1962.

[26] Dorin Maxim, Mike Houston, Luca Santinelli, Guillem Bernat, Robert I. Davis, and Liliana Cucu-Grosjean. Re-sampling for statistical timing analysis of real-time systems. In *RTNS*, 2012.

[27] C. McNairy and J. Mayfield. Montecito error protection and mitigation. In *Workshop on High Performance Computing Reliability Issues (HPCRI)*, 2005.

[28] Frank Mueller. Timing analysis for instruction caches. *Real-Time Systems - Special issue on worst-case execution-time analysis*, 2000.

[29] Jason Poovey. *Characterization of the EEMBC Benchmark Suite*. North Carolina State University, 2007.

[30] J. Reineke, D. Grund, C. Berg, and R. Wilhelm. Timing predictability of cache replacement policies. *Real-Time Systems*, 37:99–122, November 2007.

[31] D. Roberts, N. Kim, and T. Mudge. On-chip cache device scaling limits and effective fault repair techniques in future nanoscale technology. In *DSD Euromicro Conference*, 2007.

[32] RTCA and EUROCAE. *DO-178B / ED-12B, Software Considerations in Airborne Systems and Equipment Certification*, 1992.

[33] Michael Schlansker, Robert Shaw, and Sivaram Sivaramakrishnan. Randomization and associativity in the design of placement-insensitive caches. *HP Tech Report HPL-93-41*, 1993.

[34] A. Seznec and F. Bodin. Skewed-associative caches. In *PARLE*. 1993.

[35] SoCLib. -, 2003-2012. http://www.soclib.fr/trac/dev.

[36] J. Tschanz, J. Kao, S. Narendra, R. Nair, D. Antoniadis, A. Chandrakasan, and Vivek De. Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage. *IEEE Journal of Solid-State Circuits*, 37(11), 2002.

[37] F. Wartel, L. Kosmidis, C. Lo, B. Triquet, E. Quinones, J. Abella, A. Gogonel, A. Baldovin, E. Mezzetti, L. Cucu, T. Vardanega, and F.J. Cazorla. Measurement-based probabilistic timing analysis: Lessons from an integrated-modular avionics case study. In *SIES*, 2013.

[38] J. Wetzel, E. Silha, C. May, B. Frey, J. Furukawa, and G. Frazier. *PowerPC User Instruction Set Architecture*. IBM Corporation, 2005.

[39] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution time problem: overview of methods and survey of tools. *Trans. on Embedded Computing Systems*, 7(3):1–53, 2008.

[40] C. Wilkerson, H. Gao, A. Alameldeen, Z. Chishti, M. Khellah, and S.-L. Lu. Trading off cache capacity for reliability to enable low voltage operation. In *ISCA*, 2008.