Resilient Random Modulo Cache Memories for Probabilistically-Analyzable Real-Time Systems

David Trilla[†,‡], Carles Hernandez[†], Jaume Abella[†], Francisco J. Cazorla[⋆,†]
[†] Barcelona Supercomputing Center (BSC). Barcelona, Spain
[‡] Universitat Politècnica de Catalunya (UPC), Barcelona, Spain
[⋆] Spanish National Research Council (IIIA-CSIC). Barcelona, Spain.

Abstract—**Fault tolerance has often been assessed separately in safety-related real-time systems, which may lead to inefficient solutions. Recently, Measurement-Based Probabilistic Timing Analysis (MBPTA) has been proposed to estimate Worst-Case Execution Time (WCET) on high performance hardware. The intrinsic probabilistic nature of MBPTA-commpliant hardware matches perfectly with the random nature of hardware faults. Joint WCET analysis and reliability assessment has been done so far for some MBPTA-compliant designs, but not for the most promising cache design: random modulo. In this paper we perform, for the first time, an assessment of the aging-robustness of random modulo and propose new implementations preserving the key properties of random modulo, a.k.a. low critical path impact, low miss rates and MBPTA compliance, while enhancing reliability in front of aging by achieving a better – yet random – activity distribution across cache sets.**

I. Introduction

Functional and timing correctness are major concerns in the context of safety-related real-time systems. Those systems in the avionics, space, railway, medical and automotive domains, among others, provide a number of functionalities that relate to the preservation of human lives and the integrity of the system itself. It is, therefore, needed to collect enough evidence about their correct operation to certify them according to the corresponding criticality level. This relates to proving that those systems will perform their operation *correctly* and *in time*, which is assessed against the appropriate safety standards in the domain (i.e. ISO26262 in the automotive domain [19] and DO-178B/C in the avionics domain [26]).

Both concerns, functional and timing correctness, have been often faced as separated concerns. As in many other aspects, dealing with each concern separately simplifies the work but leads to inefficient solutions because the most robust design may challenge timing verification and vice versa. For instance, some recent work proposes solutions to deal with cache memory faults in the context of deterministic timing analysis, either static or measurement-based. Such timing analysis is a popular approach to estimate the Worst-Case Execution Time (WCET) of real-time tasks, needed to find a feasible schedule where all critical tasks are proven to execute before their respective deadlines [3], [4].

While deterministic timing analysis methods have been used in a wide variety of contexts, they find some non-negligible difficulties when analyzing complex programs running on top of complex high-performance hardware (e.g., cache hierarchies, multicores) [6]. Recently, an alternative timing analysis method, Measurement-Based Probabilistic Timing Analysis (MBPTA) [10] has been proposed to derive reliable WCET estimates on top of complex hardware. MBPTA relies on hardware platforms providing some properties such as random placement and replacement caches [21], [22].

Random placement and replacement caches provide a probabilistic behavior that matches very well with the random nature of hardware faults, thus facilitating the design of hardware solutions amenable for both timing analysis and fault tolerance – needed for functional correctness – simultaneously. In fact, this has been already exploited to derive probabilistic WCET (pWCET) estimates that hold valid in the presence of faults in some random placement and replacement caches [28], [29].

Recently, it has been proven that random placement may bring some performance limitations, in terms of both critical path at circuit level and cache miss rates, especially when used in first level caches. This has been addressed by proposing a new hardware design, random modulo [17], that fulfils MBPTA properties overcoming the performance issues of random placement. Random modulo randomizes cache placement within boundaries using Benes networks [8] so that high miss rate scenarios are avoided by construction, and the impact in the critical path is negligible. However, fault tolerance of random modulo has not been assessed yet.

This paper assesses fault tolerance of random modulo in terms of hot carrier injection (HCI) aging and proposes alternative random modulo implementations achieving lower HCI aging – and so higher reliability – while still preserving those properties required by MBPTA, which builds on top. The main rationale behind our designs is balancing activity across cache sets while preserving the random nature of cache placement, needed to apply MBPTA. Our results show that reliability, time analyzability and high performance can be accomodated into a single cache design outperforming existing deterministic modulo and random modulo designs.

The rest of the paper is organized as follows. Section II provides background on MBPTA and random modulo. Section III presents the reliability assessment of random modulo and our new random modulo implementations. Those different implementations are evaluated in Section IV. Section V describes some related work. Finally, Section VI draws the main conclusions of this work.

II. Background on MBPTA and Random Modulo

In this section we first introduce the fundamentals of MBPTA and the requirements it imposes on hardware design. Then we present random modulo cache design in detail since it is the basis upon which we build our work.

A. MBPTA and Its Requirements

MBPTA is a timing analysis method that derives probabilistic WCET (pWCET) estimates. Classic deterministic WCET estimation delivers a single WCET value that cannot be exceeded under any circumstance, which may lead to overly pessimistic WCET estimates [6]. Conversely, by enforcing

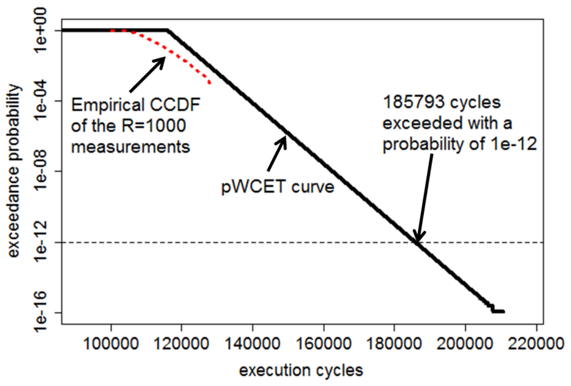

Fig. 1. Example of the application of MBPTA on 1,000 execution time measurements.

some properties on the hardware design and the way execution time measurements are collected, MBPTA [10], [9] can attach a probability of occurrence to each execution time and thus, deliver a curve that upper-bounds the execution time distribution of the program under analysis. As a result the execution time value whose exceedance probability can be deemed as irrelevant – in relation with the corresponding safety standard – is taken as WCET estimate. Note that this probabilistic WCET value is typically much lower than the actual (absolute) WCET, but exploits the fact that extremely unlikely events do not need to be upper-bounded since they fall far below the probabilities of the residual risk of the verification process as dictated by safety standards (e.g., whether all execution scenarios have been considered, whether tools have been implemented and validated properly).

In order to estimate the pWCET distribution, MBPTA relies on the application of a number of steps. Among those, we find collecting a number of execution time measurements of the program (typically in the order of some hundreds). Then, by means of Extreme Value Theory (EVT) [24], MBPTA estimates the distribution of the high execution times. Figure 1 shows the example of applying EVT to $R = 1,000$ execution time measurement (red dotted line). The plot (in logarithmic scale) shows the Complementary Cumulative Distribution Function (CCDF) also known as exceedance function. EVT delivers a function that upper-bounds the distribution of the measurements (continuous black line). For instance, for an exceedance threshold of 10^{-12} per run, the pWCET is around 186,000 cycles (continuous and dashed black lines cross), thus meaning that on average once every 10^{12} runs the execution time might be higher than 186,000 cycles.

MBPTA cannot be applied on top of arbitrary hardware designs. Instead, MBPTA needs a hardware design and a process to collect execution time measurements so that all elements introducing jitter in the execution time (e.g., cache memories) are either upper-bounded or randomized [23]. For instance, variable-latency floating point units can be made operate at their highest latency regardless of their input values. This has been done, for instance, for a FPGA-based implementation of a 4-core LEON3 processor from the Space domain where FP divisions take either 15 or 18 cycles depending on the values operated [16]. Thus, hardware has been changed to enforce always a 18-cycles latency. Other hardware resources, such as cache memories, cannot be made work at their highest latency since WCET estimates would be unaffordably high. Thus, they are time-randomized by using random placement and random replacement policies [21], which have been efficiently

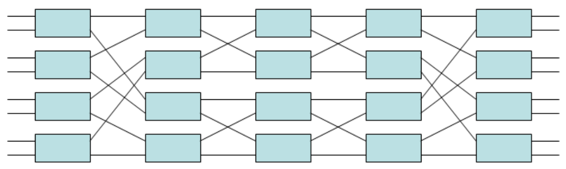

Fig. 2. Example of a 4-bit Benes network.

implemented in the same design [16] providing slightly higher average miss rates than conventional caches with modulo placement and LRU replacement [21], [22]. However, as shown recently, their impact in the critical path may be high [17] and they may produce sporadically scenarios with high miss rates even if data fits comfortably in cache [5]. These (few) problematic placements affect pWCET estimates meaningfully. Thus, random modulo [17] has been proposed recently to overcome the limitations of random placement.

B. Random Modulo Cache Design

Random placement maps each cache line randomly in a cache set based on a random seed that is changed across program runs. While this provides the properties needed by MBPTA, it may produce bad placements in terms of miss rates: even with programs accessing few cache lines, when those lines are randomly placed in the same cache set. Conversely, deterministic modulo placement maps consecutive memory lines into consecutive sets, thus avoiding this type of conflicts. However, conflicts across lines are not random and strictly depend on memory location. Since memory location of objects during operation is hard to be controlled and mimicked at analysis time, conflicts at analysis might not be representative of those during operation, hence preventing the use of MBPTA. Instead, random modulo randomizes cache placement within cache way boundaries (also using a random seed) so that, as long as cache ways do not exceed memory page size (typical case for first level caches), consecutive cache lines within cache way boundaries cannot conflict among them by construction, as it is the case for deterministic modulo placement. This is achieved by randomizing placement in the form of a random permutation. Still, conflicts among lines beyond cache way boundaries are random. Hence, average performance is close to that of deterministic modulo placement and worst-case placements deliver performance close to the average performance [17].

Random modulo is implemented by means of a Benes network where each node allows input signals to go through or to commute based on control signals (see Figure 2). In the case of random modulo, input bits (those coming from the left in the figure) are those address bits used as index in regular modulo placement, and control bits (one control bit per box, not shown in the figure) are produced by XORing conveniently the tag bits and a random seed that is changed across program executions. In this way a given memory line is randomly placed in cache, such placement holds constant during the whole execution, and it changes randomly across executions by changing the random seed (and flushing cache contents). Since addresses within cache way boundaries have distinct index bits, the Benes network delivers a bijective function so that, given specific control signals (those produced by addresses with the same tag with the same random seed) each index is placed in one set and each set corresponds to exactly one cache index. Therefore, a permutation is obtained and conflicts cannot occur across lines with identical tag.

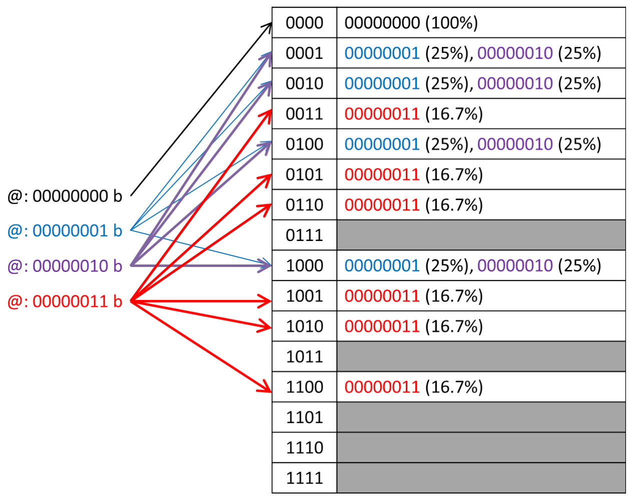

Fig. 3. Example address distribution for a 16-set cache with default random modulo design.

Note, however, that the particular way to combine tag bits and the random seed determines the particular index bit permutation chosen. Also, the output of the network is determined by the particular index bits of the address being accessed. This is detailed in the next section, where the reliability assessment is performed. In particular we show how the particular address-to-set mapping influences the utilization of each set and, therefore, their degradation in terms of HCI.

III. Enhanced Aging-Friendly Random Modulo

In this section we first describe the functioning of the default random modulo cache design, illustrating its limitations related to aging, and then we present our new enhanced random modulo implementation that makes a far more balanced use of the cache sets, thus more friendly from an aging perspective.

A. Random Modulo Set Distribution

Random modulo is intended to randomly distribute addresses (indexes in particular) to cache sets. For that purpose using a combination of address tags with random bits from the random seed to set control signals in the Benes network leads to a homogeneous and random permutation selection for index bits. However, index bits themselves are not random since they are completely program dependent.

Let us illustrate this with an example. Let us consider a program accessing 4 addresses (once offset bits have been removed): 0x00 (00000000b), 0x01 (00000001b), 0x02 (00000010b) and 0x03 (00000011b), and a cache memory with 16 sets. We realize that, regardless of the particular bit permutation selected, address 0x00 can only be placed in set 0 (0000b). Addresses 0x01 and 0x02 can be mapped instead in any set such that its set identifier contains exactly one "one", so this corresponds to sets 1 (0001b), 2 (0010b), 4 (0100b) and 8 (1000b). Any other set cannot be reached with addresses 0x01 and 0x02 regardless of the permutation selected since their lowermost 4 bits – those forming the cache index – only contain a "one". Finally, address 0x03 can only be mapped to sets 3 (0011b), 5 (0101b), 6 (0110b), 9 (1001b), 10 (1010b) and 12 (1100b) since they are the only ones whose set identifiers contain exactly 2 bits set to "one". This is graphically illustrated in Figure 3, where we can see that the amount of times each set is used is highly heterogeneous. For instance, set 0 is intensively used by address 0x00, some

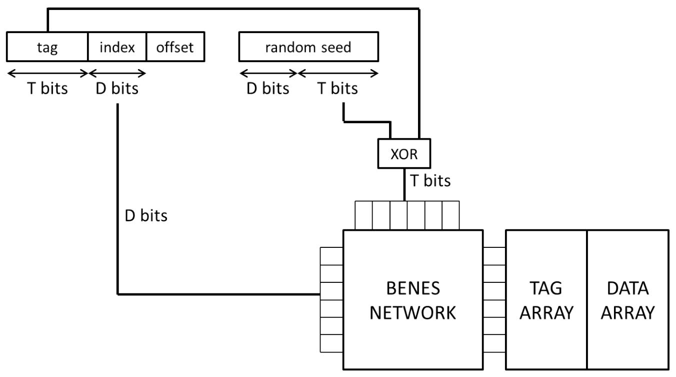

Fig. 4. Schematic of the baseline implementation of a random modulo cache.

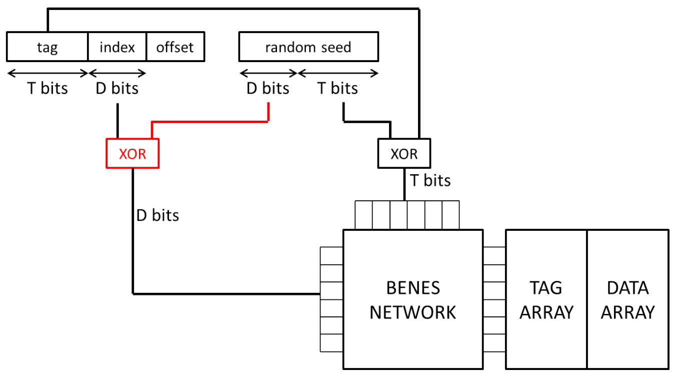

Fig. 5. Schematic of the *enhanced* implementation of a random modulo cache. Red parts indicate the changes introduced.

other sets are used with different degrees of intensity by addresses 0x01, 0x02 and 0x03, and some other sets (7, 11, 13, 14, 15) are never used since no address has enough "ones" to be mapped to any of those sets under any permutation. Further note that, if addresses are accessed heterogeneously, the impairment in the use of the different cache sets can be potentially much higher.

Having an heterogeneous cache set utilization is expected to lead to higher degradation for the most used sets due to, for instance, hot carrier injection (HCI) [31] among other sources of transistor degradation, since HCI affects devices proportionally to the activity produced, which in turn depends on the access distribution across cache sets. Hence, we aim at finding a better random modulo implementation that balances utilization of the cache sets regardless of the particular access pattern and indexes of the addresses accessed.

B. Randomizing Set Distribution

Our proposal to make index bits have a random distribution is analogous to that used for making control bits in the Benes network be random: hashing address bits with random bits. In the particular case of the index bits, we XOR them with some random bits of the random seed. For instance, let us recall our previous example. If we XOR the index bits of 0x00 (so 0000b) with a random value, in essence we will obtain 16 different indexes – all binary values that can be encoded with 4 bits – with homogeneous probability. This also holds for any other address regardless of their particular index bits. Thus, all addresses are placed to all sets with identical probability regardless of their particular index bits. Therefore, in the long run all sets are expected to be used homogeneously regardless of the particular access patterns of the programs being run.

The drawback of this approach is that a XOR gate is introduced in the path of the index bits to the Benes network, thus potentially affecting the critical path. Still, since a single XOR gate is added, the impact is limited as proven later in the evaluation section.

Figures 4 and 5 show the baseline random modulo design and our enhanced design respectively. As shown, in our enhanced version we add a level of XOR gates to combine index bits (D bits) with D random bits taken from the random seed. In the figure we show that the random bits used for the index generation (those in the left side of the Benes network) and control bits generation (those on the top part of the Benes network) correspond to different random bits. In practice there is no constraint on using the same bits or different ones since they are used for different purposes.

In summary, as shown later, this conceptually minor – but highly powerful – modification allows balancing the utilization of the cache sets, thus mitigating maximum aging and so increasing the lifetime of the cache memory. The impact in the critical path is low (at most an extra XOR gate), address-to-set mapping within cache way boundaries is a permutation (thus keeping miss rates low by avoiding many potential conflicts), and MBPTA compliance is preserved since cache set location is random.

IV. Evaluation

This section evaluates our enhanced random modulo placement. First, we introduce the evaluation methodology. Then we present the results in terms of access distribution across sets, how this improves lifetime and we show the impact in the critical path of the hardware modification.

A. Methodology

We model the first level instruction (IL1) and data (DL1) caches of a NGMP 4-core processor designed for the Space domain [7]. Those caches are 16KB 4-way 32B/line. Thus, they have 128 sets each.

We evaluate the different cache placement designs (modulo, baseline random modulo and our enhanced random modulo) with the EEMBC autobench suite, a well-known benchmark suite used in the real-time domain [25]. Each EEMBC benchmark is analyzed using the default input data for the benchmark. Considering multipath effects in the context of MBPTA has been addressed elsewhere [35] and is orthogonal to the work in this paper.

Benchmarks have been run once in an improved version of SoCLib [30] to extract instruction and address traces. Then, cache set distribution of each placement function has been evaluated in a cache simulator processing those address traces.

For estimating the lifetime improvement we use the expressions provided in [20] showing that transistors lifetime degradation due to HCI is inversely proportional to their activity. In the present work we do not consider the impact of BTI-related aging effects as they do not directly depend on the switching activity. However, we believe our proposal might also reduce the negative impact that biased cache access distributions have on BTI-related degradation. In this line, authors in [12] already pointed out that uniformly distributing accesses to cache sets also helps mitigating NBTI effects on rarely accessed sets were PMOS and NMOS devices will be stressed for a long time with biased duty cycle ratio and low switching frequency. We let as future work the quantification of lifetime improvement of our proposal when considering BTI degradation.

TABLE I
Distribution of accesses across sets for the different placement functions and caches.

Cache	**IL1**			**DL1**		
Placement	**M**	**RM**	**ERM**	**M**	**RM**	**ERM**
`a2time`	5.5	1.5	1.0	32.0	3.5	1.0
`aifftr`	3.2	2.2	1.0	17.3	2.1	1.0
`aifirf`	5.9	1.4	1.0	27.7	10.9	1.0
`aiifft`	2.4	1.5	1.0	17.2	2.1	1.0
`basefp`	7.4	1.8	1.0	36.6	4.1	1.0
`bitmnp`	2.1	1.4	1.0	26.8	2.6	1.0
`cacheb`	12.7	2.6	1.0	24.8	2.3	1.0
`canrdr`	11.1	2.0	1.0	36.4	5.7	1.0
`idctrn`	3.0	2.9	1.0	26.7	2.8	1.0
`iirflt`	7.7	1.3	1.0	20.6	4.2	1.0
`pntrch`	3.9	1.8	1.0	30.9	3.4	1.0
`puwmod`	19.1	2.7	1.0	63.3	3.2	1.0
`rspeed`	8.5	2.1	1.0	41.7	3.6	1.0
`tblook`	4.2	2.0	1.0	22.1	4.2	1.0
`ttsprk`	18.6	2.1	1.0	53.8	7.1	1.0
HARMEAN	4.9	1.8	1.0	27.9	3.4	1.0

To quantify delay overheads of the enhanced implementation of random modulo we have described both circuit implementations, the original random modulo and the enhanced one, with VHDL and synthesized them using Synopsys DC [1] with a TSMC 45nm technology library [11]. Additionally, both implementations have been integrated in a 4-core Leon3-based processor resembling the NGMP and synthesized in a Stratix IV Altera device at 100MHz.

B. Experimental Results

Set distribution. First we evaluate the access distribution across sets for each cache (IL1 and DL1) and placement function: modulo (M), random modulo (RM) and our enhanced random modulo (ERM). For each one we show the ratio between the maximum number of accesses per set and the average number of accesses per set (MAX/AVG). In the ideal case where accesses are perfectly balanced, MAX/AVG should be exactly 1. In general we can expect MAX/AVG to be higher than 1.

Table I summarizes the results for all benchmarks, with $100,000$ runs with different random seeds for RM and ERM. Since M always delivers the same distribution, one run suffices. As shown, M produces high imbalance across sets, particularly for the DL1 cache. The particular addresses accessed determine the sets accessed, and so the set distribution. Thus, M distribution is completely program-dependent. This is in part mitigated for the IL1 since loops contain some significant sequential code accessed many times, thus leading to quite homogeneous distribution (at least for the sets accessed in the loop). Conversely, access patterns for DL1 can be highly irregular in many cases, thus leading to high imbalance in the set access distribution. The harmonic mean for the set distribution of M is 4.9 for the IL1 and 27.9 for the DL1, thus far from the ideal value 1.0.

RM balances accesses much better due to the randomness introduced in the generation of the set index. This is particularly noticeable for the DL1. Still, since some dependence exists between the actual addresses accessed and the sets where they map, set distribution improves only to some extent. The harmonic mean for the set distribution of RM is 1.8 for the IL1 and 3.4 for the DL1. While these results are far better than for M, they are still far from the ideal value 1.0, especially for the DL1.

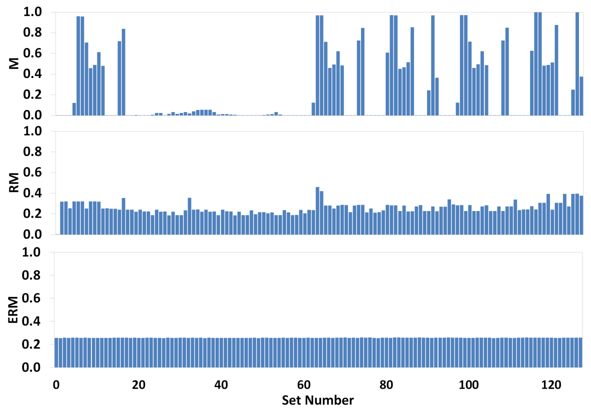

Fig. 6. IL1 per-set access distribution for `pntrch`.

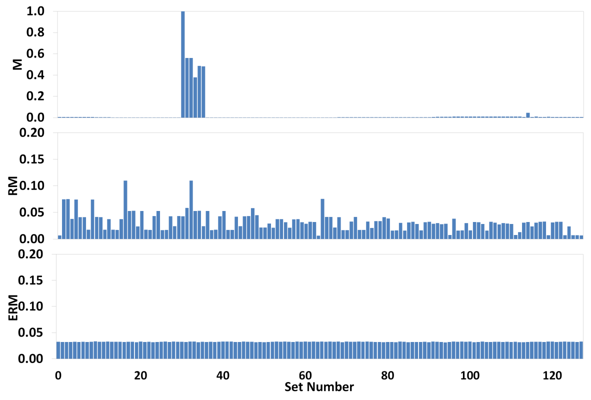

Fig. 7. DL1 per-set access distribution for `pntrch`.

Finally, our ERM removes the dependence of the set index on the particular address accessed, thus delivering much better set access distributions. This effect is particularly relevant for the DL1, where the imbalance for both M and RM is high. The harmonic mean for the set distribution of ERM is 1.0 (Max:1.028) for the IL1 and 1.0 (Max:1.044) for the DL1, very close to the ideal value 1.0.

For completeness, we show the per-set distribution for the different placement policies for 2 specific examples: the IL1 and the DL1 for `pntrch`. The former (see Figure 6) is a relatively good case for M, whereas the latter (see Figure 7) is a relatively bad case for M. Figures show in the x-axis the different cache sets and in the y-axis the utilization normalized w.r.t. the highest utilization in the M case.

The example in Figure 6 shows that M uses some sets quite often, whereas others are barely used. Still, the number of sets used often is relatively large. RM achieves a much better distribution across sets and only some sets have higher utilization than the average. Those sets correspond to those with most index bits being zero (or one), so that randomization has limited effect. Finally, our ERM achieves almost homogeneous cache set utilization. The example in Figure 7, instead, shows that M uses very few sets of the DL1 cache for `pntrch`. This leads to an extremely unbalanced distribution. In the case of RM (note that the y-axis only reaches 0.2) the distribution is far better as the most used set is used around 8 times less than for M. Still, unbalance is high. Finally, ERM achieves almost homogeneous set utilization.

Lifetime. In order to quantify the impact of the improved set distribution in lifetime, we have evaluated this effect in terms of HCI lifetime. For that purpose we use the HCI model reported in [20], as explained before, where it is shown

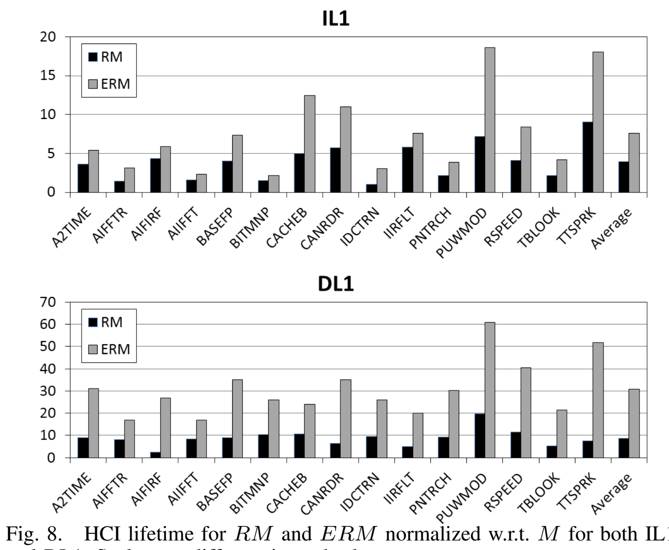

Fig. 8. HCI lifetime for RM and ERM normalized w.r.t. M for both IL1 and DL1. Scales are different in each plot.

that HCI is directly proportional to the switching activity. Thus, we can directly translate set access distribution into lifetime for HCI. We assume that a failure occurs when the first permanent fault due to HCI occurs. Thus, the cache line with highest utilization determines cache lifetime. While other models could be used, our work is centered around safety-related real-time systems where timing verification occurs before operation and assuming that the processor is fault-free. Thus, unless otherwise considered during the analysis phase, one faulty cache line may impact the WCET and thus, invalidates those timing guarantees on which the certification process has been conducted.

Since the actual lifetime value depends not only on HCI, but also on other sources of failure and hardware components, we report how much the lifetime of the IL1 and DL1 is extended due to HCI for RM and ERM normalizing the results w.r.t. M placement. Results are shown in Figure 8. We observe that lifetime grows by 3.9x and 8.8x on average for IL1 and DL1 respectively for RM. Results for ERM are far better, extending HCI lifetime by 7.6x and 30.9x on average for IL1 and DL1 respectively. If we compare ERM w.r.t. RM, lifetime grows by a factor of 1.9x and 3.5x for IL1 and DL1 respectively.

Delay impact. After synthesis we have seen ERM has zero impact w.r.t. the maximum operating frequency that the regular RM implementation can achieve. The reason is that, despite adding one level of XOR gates introduces a non-negligible impact in delay – around 0.13ns for the 45nm technology we have used – the actual critical path remains to be the one that uses TAG bits and random seed bits XORing them to configure the Benes network. Thus our modification does not impact the critical path.

V. Related Work

WCET estimation has been an important concern for both academia and industry during many years [34], [6], and it has been shown that each technique comes with its own set of features and limitations. Thus, no single technique can claim to be the best fit for all safety-related real-time systems. Recently, MBPTA [10] has been devised to obtain trustworthy and tight WCET estimates for complex software running on complex hardware where end users can only afford using measurement-based techniques. MBPTA has been successfully assessed with some industrial case studies [32], [33].

Several approaches have been proposed in the literature to address the impact of HCI and BTI in processors and cache structures [2][27]. While the majority of works focus on BTI effects, some recent works have also pointed out the importance of considering the effects of HCI degradation [18], [20], [12]. In fact, authors in [12] have already proposed improving the uniformity of accessess to the cache to mitigate HCI and NBTI degradation. Unlike in [12], where authors rely on introducing dedicated hardware resources to achieve uniform access distribution, we rely on the good properties of Random Modulo [17] to achieve the same end at an almost negligible extra cost.

Considering WCET estimation together with reliability issues has been done from many fronts. Some authors propose preserving WCET estimation methods by devising hardware cache designs able to tolerate permanent faults with no effect (or easy to account effect) on WCET estimates [3], [4]. Other authors propose accounting for the timing impact of faults in a probabilistic manner in combination with static deterministic timing analysis methods by studying the impact and probabilities of different fault distributions [13], [14], [15]. However, those approaches inherit the limitations of static timing analysis, and thus cannot be applied in the context of measurement-based timing analysis.

Recently, some authors have done some preliminary work in the context of WCET analysis of faulty hardware together with MBPTA [28], [29]. Results are promising and prove that the random nature of the timing of MBPTA-compliant hardware matches very well with the random nature of faults, thus leading to efficient solutions. However, that work considers random placement caches *as they are*. In this paper, instead, we assess the reliability of the most efficient random placement design, called random modulo [17] in terms of aging, and propose alternative random modulo implementations such that reliability is enhanced while preserving the good properties of random modulo.

VI. Conclusions

Fault tolerance and WCET estimation, needed both for safety-related real-time systems verification, have often been addressed as separate concerns. In this context, approaches based on MBPTA have been shown to match very well the needs of both concerns by relying on the same principle: randomness. Therefore, efficient solutions can be built to consider both concerns simultaneously.

In this paper we assess the reliability of random modulo cache designs, proven convenient for MBPTA, in terms of aging, and propose alternative random modulo implementations that improve aging while preserving the main features of random modulo: low impact in critical path, low miss rates and adherence to the requirements of MBPTA.

Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under the PROXIMA Project (www.proxima-project.eu), grant agreement no 611085. This work has also been partially supported by the Spanish Ministry of Science and Innovation under grant TIN2015-65316-P and the HiPEAC Network of Excellence. Jaume Abella has been partially supported by the Ministry of Economy and Competitiveness under Ramon y Cajal postdoctoral fellowship number RYC-2013-14717. Carles Hernández is jointly funded by the Spanish Ministry of Economy and Competitiveness and FEDER funds through grant TIN2014-60404-JIN.

References

[1] Synopsys design compiler. Technical report.
[2] J. Abella et al. Penelope: The NBTI-aware processor. In *MICRO*, 2007.
[3] J. Abella et al. RVC: A mechanism for time-analyzable real-time processors with faulty caches. In *HiPEAC Conference*, 2011.
[4] J. Abella et al. RVC-Based time-predictable faulty caches for safety-critical systems. In *IOLTS*, 2011.
[5] J. Abella et al. Heart of Gold: Making the improbable happen to extend coverage in probabilistic timing analysis. In *ECRTS*, 2014.
[6] J. Abella et al. WCET analysis methods: Pitfalls and challenges on their trustworthiness. In *SIES*, 2015.
[7] Aeroflex Gaisler. *Quad Core LEON4 SPARC V8 Processor - LEON4-NGMP-DRAFT - Data Sheet and Users Manual*, 2011.
[8] V. Benes. Permutation groups complexes and rearrangeable multistage connecting networks. *Bell System Technical Journal*, 43:1619–1640, 1964.
[9] F.J. Cazorla et al. Upper-bounding program execution time with extreme value theory. In *WCET Workshop*, 2013.
[10] L. Cucu-Grosjean et al. Measurement-based probabilistic timing analysis for multi-path programs. In *ECRTS*, 2012.
[11] TSMC foundry. TSMC 40 nm technology.
[12] E. Gunadi et al. Combating aging with the colt duty cycle equalizer. In *MICRO*, 2010.
[13] D. Hardy and I. Puaut. Static probabilistic worst case execution time estimation for architectures with faulty instruction caches. In *RTNS*, 2013.
[14] D. Hardy and I. Puaut. Static probabilistic worst case execution time estimation for architectures with faulty instruction caches. *Journal of Real-Time Systems*, 51(2):128–152, 2015.
[15] D. Hardy et al. Probabilistic WCET estimation in presence of hardware for mitigating the impact of permanent faults. In *DATE*, 2016.
[16] C. Hernandez et al. Towards making a LEON3 multicore compatible with probabilistic timing analysis. In *DASIA*, 2015.
[17] C. Hernandez et al. Random modulo: a new processor cache design for real-time critical systems. In *DAC*, 2016.
[18] V. Huard et al. Managing sram reliability from bitcell to library level. In *IRPS*, 2010.
[19] International Organization for Standardization. *ISO/DIS 26262. Road Vehicles – Functional Safety*, 2009.
[20] H. Kim et al. Use it or lose it: Proactive, deterministic longevity in future chip multiprocessors. *ACM Trans. Des. Autom. Electron. Syst.*, 20(4):65:1–65:26, September 2015.
[21] L. Kosmidis et al. A cache design for probabilistically analysable real-time systems. In *DATE*, 2013.
[22] L. Kosmidis et al. Multi-level unified caches for probabilistically time analysable real-time systems. In *RTSS*, 2013.
[23] L. Kosmidis et al. Probabilistic timing analysis and its impact on processor architecture. In *DSD*, 2014.
[24] S. Kotz et al. *Extreme value distributions: theory and applications*. World Scientific, 2000.
[25] Jason Poovey. *Characterization of the EEMBC Benchmark Suite*. North Carolina State University, 2007.
[26] RTCA and EUROCAE. *DO-178B / ED-12B, Software Considerations in Airborne Systems and Equipment Certification*, 1992.
[27] J. Shin et al. A proactive wearout recovery approach for exploiting microarchitectural redundancy to extend cache SRAM lifetime. In *ISCA*, 2008.
[28] M. Slijepcevic et al. DTM: Degraded test mode for fault-aware probabilistic timing analysis. In *ECRTS*, 2013.
[29] M. Slijepcevic et al. Timing verification of fault-tolerant chips for safety-critical applications in harsh environments. *IEEE Micro - Special Series on Harsh Chips*, 34(6), 2014.
[30] SoCLib. -, 2003-2012. http://www.soclib.fr/trac/dev.
[31] C.T. Wang. *Hot Carrier Design Considerations for MOS Devices and Circuits*. Van Nostrand Reinhold, 1990.
[32] F. Wartel et al. Measurement-based probabilistic timing analysis: Lessons from an integrated-modular avionics case study. In *SIES*, 2013.
[33] F. Wartel et al. Timing analysis of an avionics case study on complex hardware/software platforms. In *DATE*, 2015.
[34] R. Wilhelm et al. The worst-case execution time problem: overview of methods and survey of tools. *ACM TECS*, 7(3):1–53, 2008.
[35] M. Ziccardi et al. EPC: Extended path coverage for measurement-based probabilistic timing analysis. In *RTSS*, 2015.