# Probabilistic WCET estimation in presence of hardware for mitigating the impact of permanent faults

Damien Hardy, Isabelle Puaut, Yiannakis Sazeides

# Probabilistic WCET estimation in presence of hardware for mitigating the impact of permanent faults

Damien Hardy
University of Rennes 1/IRISA
damien.hardy@irisa.fr

Isabelle Puaut
University of Rennes 1/IRISA
isabelle.puaut@irisa.fr

Yiannakis Sazeides
University of Cyprus
yanos@cs.ucy.ac.cy

***Abstract*—Fine-grained disabling and reconfiguration of hardware elements (functional units, cache blocks) will become economically necessary to recover from permanent failures, whose rate is expected to increase dramatically in the near future. This fine-grained disabling will lead to degraded performance as compared to a fault-free execution.**

**Until recently, all static worst-case execution time (WCET) estimations methods were assuming fault-free processors, resulting in unsafe estimates in the presence of faults. The first static WCET estimation technique dealing with the presence of permanent faults in instruction caches was proposed in [1]. This study probabilistically quantified the impact of permanent faults on WCET estimates. It demonstrated that the probabilistic WCET (pWCET) estimates of tasks increase rapidly with the probability of faults as compared to fault-free WCET estimates. In this paper, we show that very simple reliability mechanisms allow mitigating the impact of faulty cache blocks on pWCETs. Two mechanisms, that make part of the cache resilient to faults are analyzed. Experiments show that the gain in pWCET for these two mechanisms are on average $48\%$ and $40\%$ as compared to an architecture with no reliability mechanism.**

## I. Introduction

Safety-critical systems require that their deadlines are met in all execution situations, including the worst-case. This proof needs an estimation of the worst-case execution time (WCET) of any sequential task in the system. WCET estimation methods [2] have to be *safe* and *tight* at the same time. *Safety* is the guarantee that estimated WCETs are greater than or equal to any possible execution time. *Tightness* means that the estimated WCET is as close as possible to the actual WCET (which in general is unknown). Static WCET estimation methods are known to provide safe WCET estimates if conservative information on the timing of instructions is available.

A common implicit assumption in a large majority of static WCET estimation techniques is that the hardware is not subject to faults. However, technology scaling, used to increase performance, has the negative consequence of providing less reliable silicon primitives due to static and dynamic variations [3]. This results in an increase of the probability of failure (*pfail*) of circuits. The resilience roadmap published in [4] underlines the magnitude of the problem we will very soon face. According to [4], the increase of *pfail* with scaling will be particularly significant for SRAM cells, for which the predicted *pfail* is $6.1 \times 10^{-13}$ at 45nm and will increase to $2.6 \times 10^{-04}$ at 12nm. Furthermore, when using dynamic voltage and frequency scaling (DVFS), SRAM cells may begin to fail if the voltage is reduced too much. In [5] the predicted *pfail* for 32nm technology is $1 \times 10^{-3}$ at 0.5V. Existing techniques defined to deal with process and latent defects and provide fault-free chips at current technology nodes, like column/row sparing and error correcting codes (ECC), will not be affordable anymore in the future due to their non-scalable cost when used extensively to recover from permanent faults. Consequently, other approaches like fine-grained disabling and reconfiguration (e.g. of individual functional units or cache blocks) will become economically necessary. We are going to enter a new era: functionally correct chips with variable performance among a population of chips, and with varying performance over time resulting from aging. A recent study [6] has analyzed the effect of fine-grained disabling resulting from permanent faults on average performance. It reveals that caches, which take most of the die real-estate in current processors and contain numerous SRAM cells, will be a non-negligible source of performance degradation in the near future. Therefore, assuming fault-free chips in WCET estimation methods may lead to an underestimation of the WCET, even when using static analysis methods.

In a previous study [1], we have proposed a static probabilistic method to estimate the impact of permanent faults in cache blocks on worst-case execution times. The proposed method statically estimates the worst-case path in programs using static analysis, its probabilistic nature only stemming from the probabilistic nature of cell failure. Derived probabilistic WCETs (pWCETs in the following) can be used to ensure that timing constraints are met under a targeted probability depending on the software criticality level (e.g. $10^{-15}$ per task activation for the commercial aerospace industry [7]).

The experimental evaluation of [1] allowed to quantify the impact of permanent faults on pWCETs. For a given probability of cell failure (e.g. $10^{-4}$), it was observed that the pWCET estimates are significantly higher than fault-free WCET estimates. There is thus a need to mitigate the impact of faults to significantly reduce pWCET estimates. An in depth observation reveals that when the targeted probability is very low (e.g. $10^{-15}$) some sets are considered as entirely faulty. This leads to a significant degradation of pWCET estimates, because spatial locality in faulty sets is not captured anymore.

In this paper, we show that simple reliability mechanisms allow to eliminate this situation. This additional hardware

allows to obtain pWCET estimates that are much lower than in [1], and in some situations identical to those of a fault-free architecture. More precisely, the contributions of this paper are the following:

- Probabilistic WCET estimation techniques accounting for permanently faulty cache blocks for two reliability mechanisms. The two mechanisms differ by their hardware cost and impact on estimated pWCETs, to allow the hardware designer to find the best pWCET/cost tradeoff;
- An experimental evaluation of the impact of the considered reliability mechanisms on estimated pWCETs. Experimental results show that the gain in pWCET for the two mechanisms are on average $48\%$ and $40\%$ as compared to an architecture with no reliability mechanism.

The remainder of the paper is organized as follows. The fault model and the base probabilistic fault-aware WCET estimation technique are presented in Section II. Section III describes the two reliability mechanisms to support faulty cache blocks, and presents the corresponding pWCET estimation. Experimental results are given in Section IV. Section V surveys related work. Conclusions are given in Section VI.

## II. Background

In this section we present the considered architecture and fault model. We also detail the used state-of-the-art WCET estimation technique. Finally, we give an overview of our previous work on probabilistic WCET estimation for faulty instruction caches [1] that will serve as a base in this paper.

### A. Architecture and fault model

The analysis is defined for set-associative instruction caches implementing the Least Recently Used (LRU) replacement policy. A cache configuration is defined by a number of sets $S$, a number of ways per set $W$, and a block size in bits $K$. For the scope of the paper, a single level of instruction cache is assumed.

The study assumes that permanent faults manifest only in the instruction cache. The extension of this work to other processor structures is subject of other ongoing effort.

A cache block with at least one bit affected by a permanent fault is considered as faulty and is disabled. Furthermore, LRU-stack bits and control bits are assumed to be fault free.

Faulty cache blocks are assumed to be detected using post-manufacturing and boot-time tests, ECC, and built-in self-tests, and are disabled when detected as faulty. The exact position of the faulty blocks in each set has no importance, thanks to LRU replacement: in case of block failure, the size of the LRU stack of a set is reduced by its number of faulty blocks. Each SRAM cell (bit) is assumed to have an equal probability of failure *pfail* to be permanently faulty. The locations of permanently faulty SRAM cells are considered as random. This allows to account for major causes of uncorrelated faults [8].

Based on the above assumptions, with $K$ the block size in bits, the probability of a cache block failure $p_{bf}$ can be determined from the probability of bit failure *pfail* as follows:

$$p_{bf} = 1 - (1 - pfail)^K \tag{1}$$

The probability $p_{wf}(w)$ to have exactly $w$ faulty ways among $W$ in a set can be derived by using the binomial probability law:

$$p_{wf}(w) = \binom{W}{w} (p_{bf})^w (1 - p_{bf})^{W-w} \tag{2}$$

When reliability mechanisms will be introduced in Section III, the faults concerning hardened blocks will be masked. In the following, the term *faulty block* will denote a block affected by a fault and not covered by the reliability mechanisms.

### B. Static WCET estimation technique

Static WCET estimation techniques consist of a *low-level* analysis to analyze the timing of instructions, and a *high-level* analysis to identify the longest execution path. The state-of-the-art techniques we use are presented below. Since our focus is on the impact of permanent faults on instruction caches, the description of the low-level analysis only focuses on that hardware component.

*1) Instruction cache analysis:* The contribution of instruction caches to the WCET is determined by statically associating a Cache Hit/Miss Classification (CHMC) for every memory reference. The CHMC defines for each reference its worst-case behavior with respect to the instruction cache (e.g. the CHMC is set to *always-hit* only when it is guaranteed the reference will always result in a cache hit, regardless of the execution path followed at run-time).

CHMCs are obtained by applying the static analysis technique described in [9], based on abstract interpretation. Three analyses that operate on the program control flow graph are defined. The *Must* analysis determines if a memory block is *always* present in the cache at a given program point: if so, the reference CHMC is *always-hit*. The *Persistence* analysis determines if a memory block will not be evicted after it has been loaded; the CHMC of such references is *first-miss*. Finally, the *May* analysis determines if a reference to a memory block may be in the cache at a given point: if not, the reference CHMC is *always-miss*. Otherwise, if present neither in the Must analysis nor in the Persistence analysis the reference CHMC is *not classified*.

Each analysis computes an *Abstract Cache State* (ACS) at every program point until a fixpoint is reached. The semantics of abstract cache states depends on the considered analysis. All of them maintain for each cache block an *age* in the LRU stack (the smallest age is the block in the Most Recently Used - MRU position, the highest age is the Least Recently Used block). For the *Must* and *Persistence* analyses, the age of a block in an ACS is the maximum possible age the block could have in the cache at run-time.

*2) High-level analysis:* WCET calculation uses the most prevalent technique, named IPET for *Implicit Path Enumeration Technique*. IPET is based on an Integer Linear Programming (ILP) formulation of the WCET calculation problem [10]. It reflects the program structure and the possible execution flows using a set of linear constraints. Such constraints express that each basic block must be entered the same number of times as it is exited (structural constraints), or indicate the maximal number of times a basic block can be executed inside loops (loop bound constraints).

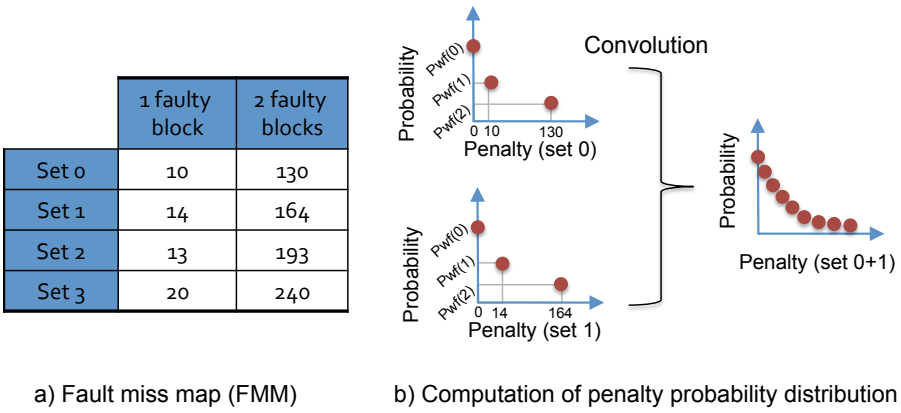


Fig. 1. (a) Fault miss map (FMM). (b) Example of computation of the distribution of fault-induced miss penalties for the first two sets.

### C. Probabilistic fault-aware WCET estimation

The technique described in [1] computes, for a given program, cache configuration, and probability of SRAM cell failure, a WCET probability distribution, representing the WCET estimate probability among a population of chips. In order to avoid an exhaustive computation of WCETs for all combinations of fault locations, the approach operates in two steps. It first computes the fault-free (non probabilistic) WCET as described in section II-B. Then, it determines an upper bound of the probability distribution of the time penalties caused by fault-induced misses. The WCET probability distribution is then obtained by adding these two components.

The penalty probability distribution is computed based on a *Fault Miss Map* (FMM). In this map (Figure 1.a), each row corresponds to a set and each column corresponds to a number of faulty blocks in the set. Each FMM element gives an upper bound of the number of fault-induced misses per set and per number of faulty blocks. For example, in the FMM of Figure 1.a, we know that a single faulty block in set 0 results in at most 10 additional cache misses. The FMM is computed by solving for each set $s$ and each number of faulty blocks $f$ an ILP system close to IPET (see [1] for details).

The penalty distribution is computed for each cache set independently, by studying the respective impact of $f \in [1, W]$ faults. The distribution is composed of at most $W+1$ different penalties that correspond to 0 up to $W$ faulty blocks with the probability for each case given by $p_{wf}$ (equation 2). Since sets are independent, all these discrete distributions are independent. Thus, they are combined by computing the product (convolution) of their probability distributions to obtain an upper bound of the penalty distribution, as illustrated in Figure 1.b. In the figure, the penalty probability distribution of set 0 and set 1 have 3 distinct points corresponding to 0, 1 and 2 faulty blocks, with their associated probability to occur $p_{wf}(0)$, $p_{wf}(1)$ and $p_{wf}(2)$. These two distributions are then combined by convolution to obtain the penalty probability distribution of set $0+1$.

## III. Probabilistic WCET estimation in presence of reliable hardware

In this section we first present two independent reliability mechanisms, whose objective is to mitigate the impact of permanent faults on probabilistic WCET (pWCET) estimates. Then, we show how pWCETs are estimated in the presence of such reliability mechanisms.

### A. Hardware to support faulty cache blocks

*1) Reliable way (RW):* The first hardware modification, named *RW* in the following, is to have one fixed way per set (for example way 0) resilient to permanent faults [11], [12]. Figure 2.a gives an illustration of a cache with the *RW* mechanism. The idea behind this mechanism is to cope with the situations when entire sets are faulty. Without this mechanism, a significant increase of fault-induced misses would arise, because the spatial locality of references would not be captured. With the added hardware, the spatial locality is always captured regardless of faults, as well as a part of temporal locality of references. Making one way in the cache resilient to faults ensures that the performance will be at worst the one of a direct-mapped cache of size $S$.

*2) Shared Reliable buffer (SRB):* The second hardware extension is the addition of a small shared buffer (same size as a L1 cache block), named *SRB* designed such that it is resilient to permanent faults (see Figure 2.b).

In contrast to the *RW*, the cache look-up mechanism is modified. When a given address is referenced, the memory block is searched for into the *SRB only* when all blocks in the concerned set are faulty; in case of a miss in the *SRB*, the missing memory block is loaded into the *SRB*. If the corresponding set has at least one non-faulty cache block, the *SRB* is not used and the cache look-up is done as usual.

Similarly to the *RW*, the idea of the *SRB* is to avoid the drastic increase of cache misses occurring when all cache blocks in a set are faulty, but with a comparatively lower hardware cost. However, compared to the *RW*, the *SRB* may result in worse performance in the presence of faults, because the size of the reliable memory is lower than for the *RW*.

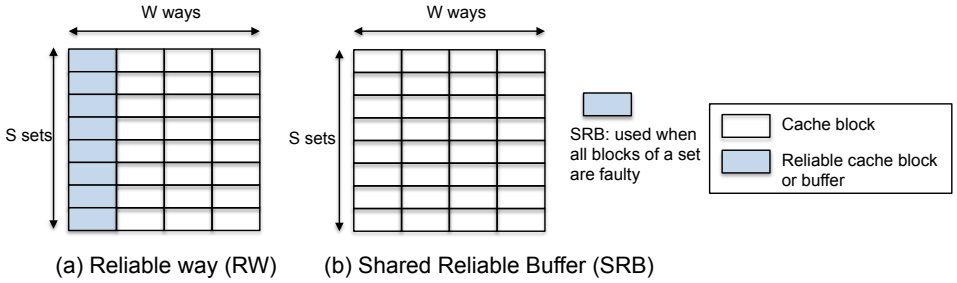


Fig. 2. Hardware to mask faulty cache blocks

Moreover, the *RW* is more likely to benefit from temporal locality than the *SRB*.

### B. Probabilistic WCET estimation with the RW and SRB

*1) RW pWCET estimation:* The *RW* ensures that at most $W-1$ ways are faulty. A fault may affect the reliable way with a probability $p_{bf}$ as defined before, but should this be the case, the fault will be masked. Thus, when computing the penalty distribution of a set, $W-1$ ways only have to be considered and the fault-induced misses when all blocks are faulty can be safely ignored.

Formally, equation 2 is modified as follows:

$$p_{wf}(w) = \binom{W-1}{w} (p_{bf})^w (1-p_{bf})^{W-1-w} \qquad (3)$$

Intuitively, the *RW* allows to remove from the penalty distribution of each set, the point corresponding to $W$ faulty blocks, leading to the highest penalty. For instance, the penalty of set 0 in Figure 1.b will have only two points instead of three: one with a penalty of 0 and one with a penalty of 10.

*2) SRB pWCET estimation:* When using the *SRB*, fault-induced misses in the MRU position can still occur since the *SRB* is shared by all sets and thus is mainly able to preserve the spatial locality.

The analysis of a cache with a *SRB* requires a modification of the computation of the FMM to remove all the fault-induced misses that are for sure eliminated by the *SRB*. To do so, a cache analysis of the *SRB* is performed as if the *SRB* was the only cache in the system. This allows to capture the spatial locality of references within the *SRB*. Then, when computing an upper bound of fault-induced misses when all blocks are faulty in a set, the references classified as *always-hit* in the *SRB* can be safely removed. The penalty distribution is finally computed as before. Intuitively, the *SRB* mechanism allows to remove most (but not all) of the fault-induced misses in the rightmost column of the FMM. Thus, the point with the highest penalty in the penalty distribution of each set will be significantly reduced.

Remark that our analysis conservatively assumes that no information is kept in the *SRB* between distinct series of successive accesses to the *SRB*. For example, in the reference stream $a_1a_2b_1b_2a_1a_2$, if addresses $a_i$ and $b_i$ are mapped to distinct sets, the cache analysis of the *SRB* will classify all occurrences of $a_2$ and $b_2$ as *always-hit*. The second occurrence of $a_1$ will conservatively be classified as *not classified* because there is a non-null probability that the *SRB* is reloaded when referencing $b_1$ in case the corresponding set is faulty. The definition of a more precise analysis deriving the probability that a block stays in the *SRB* is left for future work.

## IV. Experimental results

### A. Experimental setup

The experiments were conducted on 25 benchmarks from the Mälardalen WCET benchmark suite [13]. The instruction cache size is fixed to 1KB. The cache and memory latencies are fixed to 1 cycle and 100 cycles respectively. The cache configuration is fixed to a 4-ways set-associative cache with LRU replacement and with 16B cache lines. This configuration is the one leading to the smallest pWCET in [1].

The experiments were conducted on MIPS R2000/R3000 binary code compiled with gcc 4.1 with no optimization and the default linker memory layout. Fault-free task WCETs are computed by the Heptane static WCET estimation tool [14], that implements the cache analysis and WCET computation steps as presented in Section II-B. The experimental results presented hereafter only account for the contribution of instruction caches to the WCET. The effects of other architectural features (data cache, pipeline, branch predictor) are not considered. Cache Hit/Miss Classification *not classified* is considered equivalent to *always-miss*. Cplex 12.5 is used to solve the different ILP systems.

The probability of bit failure *pfail* is fixed to $10^{-4}$ which is representative of the highest assumed probability of cell failure in related work [4], [7]. That *pfail* value helps identify the different behaviors that can be observed in our experiments.

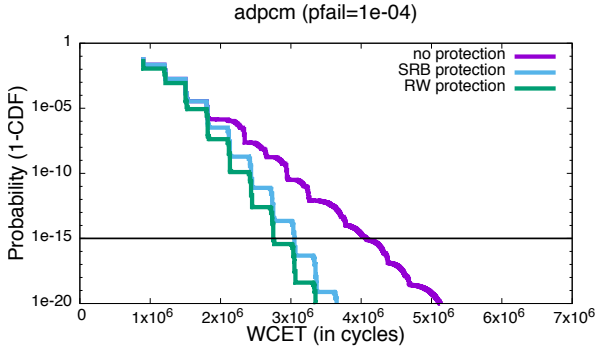


Fig. 3. Complementary cumulative distribution with the *SRB*, the *RW* and *no protection* for benchmark *adpcm*, for $pfail = 10^{-4}$.

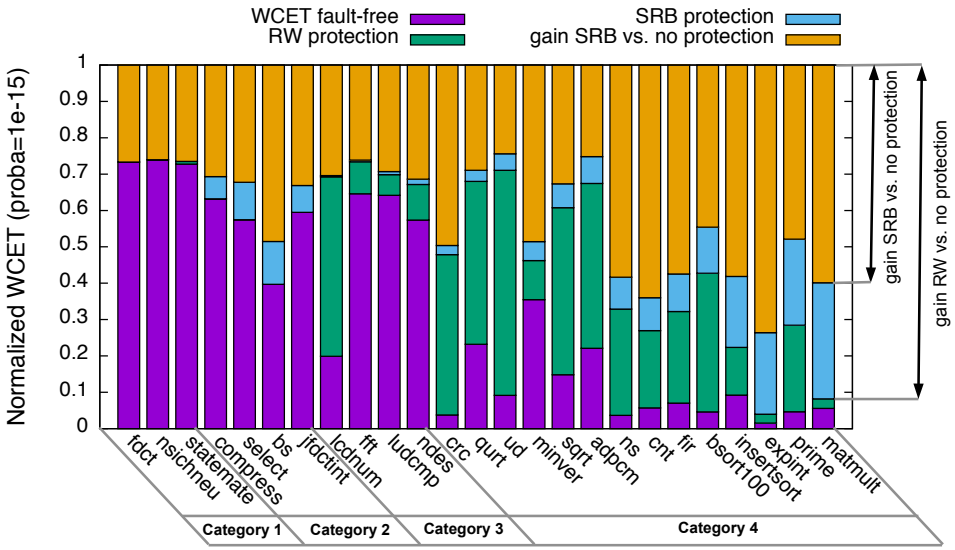


Fig. 4. pWCET estimates for a fault-free architecture, an architecture with the *SRB* and an architecture with the *RW*, all normalized against the pWCET on a system with no protection. The targeted probability is set to $10^{-15}$ ($pfail = 10^{-4}$).

The analyses provide complementary cumulative distribution functions, as illustrated in Figure 3, which depicts the pWCET of benchmark *adpcm* for the different levels of protection (no protection, *RW* and *SRB*). The three curves are exceedance functions indicating for a given targeted probability $p$ (e.g. $10^{-15}$ per task activation for aerospace commercial industry [7]) the value at which the random variable WCET will be equal to, or below that value, with probability $1 - p$.

For brevity the results in the following will be presented by depicting only the pWCET estimates obtained when the targeted probability is set to $10^{-15}$ instead of giving the complete complementary cumulative distribution.

### B. Experimental results

Figure 4 reports the pWCET estimates for three configurations: a fault-free architecture, an architecture with the *RW* and an architecture with the *SRB*. Values are normalized with respect to the pWCET estimated without any protection mechanism. The benefit of the *SRB* mechanism is highlighted by the top part of the stacked histogram, while for the *RW* the benefit is observed by cumulating the benefit of the *SRB* and the contribution of the *SRB* (see the illustration on *matmult* at the right of the figure). In Figure 4, benchmarks having similar behavior are grouped together.

In the figure, we can observe that for all benchmarks, using the *SRB* or the *RW* results in significantly lower pWCETs compared to an architecture with no protection. For the *SRB*, the minimum gain is $25\%$ for benchmark *ud* and $40\%$ on average. The gain for the *RW* is as expected larger than or equal to the gain with the *SRB*, with a minimum gain of $26\%$ for benchmark *fft* and an average gain of $48\%$.

A deeper analysis of the results, for all benchmarks, reveals that the benchmarks can be classified in four different categories, depicted in Figure 4 from left to right, depending on the locality captured by the cache.

For the first category, the estimated pWCET for both the *RW* and the *SRB* is equal to the fault-free WCET. It means that the impact of faults is fully masked by the protection mechanisms. This is explained by the fact that the cache, because of the applications' characteristics, is able to capture the spatial locality only, which is fully preserved by the two mechanisms.

For the second category, the estimated pWCET for the *RW* is equal to the fault-free WCET but not the pWCET for the *SRB*. For these benchmarks, the cache is able to capture only the spatial and temporal locality in the MRU position. Since the *SRB* analysis is not able to preserve the temporal locality in the MRU position, this explains why the gain of the *SRB* is reduced as compared to the *RW*.

The third category shows a similar gain for both mechanisms. This is explained by the fact that most of the temporal locality captured by the cache is not covered by the MRU position and thus cannot be protected by the protection mechanisms.

The fourth and last category mixes the behaviors of the two previous categories: $(i)$ the temporal locality captured by the MRU position and $(ii)$ the temporal locality captured by the rest of the cache. This explains the difference between the two proposed mechanisms and between them and the fault-free WCET.

The different impact of the two mechanisms on pWCETs is expected since the *SRB* mainly captures spatial locality and our analysis of the *SRB* is conservative. In contrast, the *RW* better takes benefit of the temporal locality, but has a higher hardware cost. As shown in Figure 4, the tradeoff between the two protection mechanisms does not only depend on the extra hardware cost but also on the applications characteristics.

## V. Related Work

WCET estimation techniques may use either *static analysis* or *measurements* to derive WCET bounds [2]. Our method

falls into the first category, which by construction provides safe WCET estimates. Moreover, produced WCET estimates may be either *deterministic* (calculate a strict upper bound that is never exceeded at run-time) or *probabilistic* (calculate several worst-case execution times with associated probabilities of occurrence). The method we propose in this paper can be classified as *static probabilistic timing analysis* (SPTA). The use of static analysis guarantees the longest execution path is detected; the method produces *probabilistic* WCET bounds to reflect the probabilistic nature of faults.

Many static timing analysis methods for systems equipped with cache memories (instruction caches, data caches, cache hierarchies, with various cache structures and replacement policies) have been proposed in the past [9], [15], [16], [17]. To the best of our knowledge, all of them, except [1] assume fault-free caches. The added contribution of this paper as compared to [1] is to add cost-effective hardware to improve the pWCET estimates in the presence of faults.

The impact of faults on software timing was studied in a series of research papers [6], [18], [7]. Paper [6] addresses the impact of permanent faults and disabling of resources on application performance for non real-time applications. They concentrate on average-case performance instead of worst-case performance, and do not propose any additional hardware to limit the impact of faults on performance. A method to statically compute probabilistic WCETs in the presence of faults is presented in [18]. In contrast to our work, [18] focuses on faulty sensors, whereas we focus on faulty SRAM cells in instruction caches. The objective of [7] is to derive WCET estimates in the presence of permanent faults and disabling of hardware elements. The difference lies in the type of method used to obtain WCETs and in the processor architecture considered. Regarding WCET estimation, [7] uses a *measurement-based probabilistic timing analysis* method and as such is not guaranteed to identify the longest execution path, whereas we use static probabilistic timing analysis (SPTA).

Previous studies have proposed new hardware to limit the impact of permanent faults on performance. A reliable victim cache (RVC) is proposed in [19]; it allows replacing faulty cache lines with supplementary cache lines from the victim cache. The authors of [19] evaluate the impact of the RVC on WCETs using cycle-accurate simulation along the already known worst-case execution path; in contrast to our work, they do not provide means to identify the longest execution path. Cost-effective approaches consisting of a spectrum of cell sizes are proposed in [11], [20] to limit the impact of failures occurring at low voltage in caches, in architectures using dynamic voltage/frequency scaling. In contrast to our work, the focus in [11], [20] is on average-case performance.

## VI. Conclusion

We have proposed in this paper a pWCET analysis for two reliability mechanisms (*SRB* and *RW*) that limit the impact of permanently faulty cache blocks on worst-case performance. As future work, a more precise pWCET estimation technique for the *SRB* could be devised to limit the conservatism of the proposed technique. Other directions are to transpose the hardware and corresponding analyses to data caches, and to perform an extensive analysis of the impact of the proposed mechanisms on die area and power consumption.

## Acknowledgments

This work has been partially supported by the COST Action IC1202: Timing Analysis On Code-Level (TACLe). The authors would like to thank André Seznec and Pierre Michaud for their fruitful comments on earlier versions of this paper.

## References

[1] D. Hardy and I. Puaut, "Static probabilistic worst case execution time estimation for architectures with faulty instruction caches," *Real-Time Systems*, vol. 51, no. 2, pp. 128–152, 2015.

[2] R. Wilhelm *et al.*, "The worst-case execution-time problem-overview of methods and survey of tools," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, pp. 36:1–36:53, May 2008.

[3] S. Borkar *et al.*, "Parameter variations and impact on circuits and microarchitecture," in *DAC40*, Jun. 2003, pp. 338–342.

[4] S. R. Nassif *et al.*, "A resilience roadmap," in *DATE*, 2010, pp. 1011–1016.

[5] S.-T. Zhou *et al.*, "Minimizing total area of low-voltage SRAM arrays through joint optimization of cell size, redundancy, and ECC," in *IEEE International Conference on Computer Design*, oct. 2010, pp. 112 –117.

[6] D. Hardy *et al.*, "The performance vulnerability of architectural and non-architectural arrays to permanent faults," in *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012, pp. 48–59.

[7] M. Slijepcevic *et al.*, "DTM: Degraded test mode for fault-aware probabilistic timing analysis," in *25th Euromicro Conference on Real-Time Systems*, 2013, pp. 237–248.

[8] L. Cheng *et al.*, "Physically justifiable die-level modeling of spatial variation in view of systematic across wafer variability," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 30, no. 3, pp. 388–401, 2011.

[9] H. Theiling *et al.*, "Fast and precise WCET prediction by separated cache and path analyses," *Real-Time Systems*, vol. 18, no. 2-3, pp. 157–179, 2000.

[10] Y.-T. S. Li and S. Malik, "Performance analysis of embedded software using implicit path enumeration," in *Proceedings of the 32Nd Annual ACM/IEEE Design Automation Conference*, 1995, pp. 456–461.

[11] H. R. Ghasemi *et al.*, "Low-voltage on-chip cache architecture using heterogeneous cell sizes for high-performance processors," in *17th International Conference on High-Performance Computer Architecture, February 12-16 2011, San Antonio, Texas, USA*, 2011, pp. 38–49.

[12] J. Kulkarni *et al.*, "A 160 mv robust schmitt trigger based subthreshold sram," *Solid-State Circuits, IEEE Journal of*, vol. 42, no. 10, pp. 2303–2313, Oct 2007.

[13] J. Gustafsson *et al.*, "The Mälardalen WCET benchmarks: Past, present and future." *10th International Workshop on Worst-Case Execution Time Analysis*, vol. 15, pp. 136–146, 2010.

[14] A. Colin and I. Puaut, "A modular and retargetable framework for tree-based WCET analysis," in *Euromicro Conference on Real-Time Systems (ECRTS)*, Delft, The Netherlands, Jun. 2001, pp. 37–44.

[15] F. Mueller, "Timing analysis for instruction caches," *Real-Time Systems*, vol. 18, no. 2-3, pp. 217–247, 2000.

[16] C. Ferdinand and R. Wilhelm, "On predicting data cache behavior for real-time systems," in *Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems*, 1998, pp. 16–30.

[17] D. Hardy and I. Puaut, "WCET analysis of multi-level non-inclusive set-associative instruction caches," in *Proceedings of the 29th Real-Time Systems Symposium*, Dec. 2008, pp. 456–466.

[18] K. Höfig, "Failure-dependent timing analysis - a new methodology for probabilistic worst-case execution time analysis," in *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*, ser. Lecture Notes in Computer Science, J. Schmitt, Ed., vol. 7201, 2012, pp. 61–75.

[19] J. Abella *et al.*, "RVC: A mechanism for time-analyzable real-time processors with faulty caches," in *Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers*, 2011, pp. 97–106.

[20] D. Palframan *et al.*, "iPatch: Intelligent fault patching to improve energy efficiency," in *IEEE 21st International Symposium on High Performance Computer Architecture*, Feb 2015, pp. 428–438.