Real-Time Syst (2015) 51:128–152
DOI 10.1007/s11241-014-9212-x

Static probabilistic worst case execution time estimation for architectures with faulty instruction caches

Damien Hardy · Isabelle Puaut

Published online: 5 November 2014

Abstract Semiconductor technology evolution suggests that permanent failure rates will increase dramatically with scaling, in particular for SRAM cells. While well known approaches such as error correcting codes exist to recover from failures and provide fault-free chips, they will not be affordable anymore in the future due to their growing cost. Consequently, other approaches like fine-grained disabling and reconfiguration of hardware elements (e.g. individual functional units or cache blocks) will become economically necessary. This fine-grained disabling will degrade performance compared to a fault-free execution. To the best of our knowledge, all static worst-case execution time (WCET) estimation methods assume fault-free processors. Their result is not safe anymore when fine-grained disabling of hardware components is used. In this paper we provide the first method that statically calculates a probabilistic WCET bound in the presence of permanent faults in instruction caches. The proposed method derives a probabilistic WCET bound for a program, cache configuration, and probability of cell failure. As our method relies on static analysis to bound the longest path, its probabilistic nature only stems from the probability that faults actually occur. Our method is computationally tractable because it does not require an exhaustive enumeration of all the possible combinations of faulty cache blocks. Experimental results show that it provides WCET estimates very close to, but never below, the method that derives probabilistic WCETs by enumerating all possible locations of faulty cache blocks. The proposed method not only allows to quantify the impact of permanent faults on WCET estimates, but, most importantly, can be used in architectural explo-

D. Hardy (✉) · I. Puaut
University of Rennes 1/IRISA, Rennes, France
e-mail: damien.hardy@irisa.fr

I. Puaut
e-mail: isabelle.puaut@irisa.fr

ration frameworks to select the most appropriate fault management mechanisms and design parameters for current and future chip designs.

Keywords Hard real-time · WCET · Static probabilistic analysis · Faulty caches

1 Introduction

In safety-critical real-time systems it is crucial to prove that tasks meet their deadline 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 must offer both *safety* and *tightness* properties. *Safety* is the guarantee that the estimated WCET is greater than or equal to any possible execution time. *Tightness* means that the estimated WCET is as close as possible to the actual WCET.

WCET estimation methods can be divided into two classes (Wilhelm et al. 2008): *measurement-based* approaches and *static analysis* approaches. The former class of methods executes the task under study to derive a WCET estimate. It has the advantage to provide an estimation when the processor micro-architecture is not completely known. On the other hand it may underestimate the actual WCET if test-case generation does not cover the actual longest path during measurements, and thus may raise safety issues. The latter class of methods, in contrast, statically analyzes the software and hardware under study. It provides a safe WCET upper bound but requires detailed knowledge of the processor micro-architecture.

Static WCET estimation methods are generally divided into two steps, commonly named *high-level* analysis and *low-level* analysis. The *high-level* analysis determines the longest execution path among all possible flows in a program. The *low-level* analysis is used to account for the processor micro-architecture. A number of static analysis methods have been designed in the last two decades at both levels (see Wilhelm et al. 2008 for an extensive survey of methods and tools). Concerning the low-level analysis, methods have been designed to predict WCETs in architectures equipped with caches (Theiling et al. 2000; Mueller 2000; Ferdinand and Wilhelm 1998; Hardy and Puaut 2008) or pipelines (Engblom 2002; Li et al. 2006).

A common implicit assumption in all static WCET estimation methods is that the target processor 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 (Borkar et al. 2003; Bowman et al. 2009), resulting in an increase of the probability of failure (*pfail*) of circuits. The recently published resilience roadmap (Nassif et al. 2010) underlines the magnitude of the problem we will very soon face. According to Nassif et al. (2010), the increase of *pfail* with scaling will be particularly significant for SRAM cells, for which the predicted *pfail* is 6.1×10^{-13} at 45 nm and will increase to 2.6×10^{-4} at 12 nm.

Space redundancy techniques, like column/row sparing and error correcting codes (ECC), have been defined to deal with process defects and latent errors and provide fault-free chips at current technology nodes. They will not be affordable anymore in the future due to their non-scalable cost, when used extensively to recover from permanent faults. Furthermore, using ECC to prevent the effect of permanent faults will reduce the transient errors recovery capabilities.

Consequently, other approaches such as fine-grained disabling and reconfiguration (e.g. 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 chips and throughout their lifetime. A recent study (Hardy et al. 2012) has analyzed the effect of fine-grained disabling on average performance resulting from permanent faults. 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, ignoring the effect of such faults in WCET estimation methods may lead to an underestimation of the WCET, even for static analysis methods.

To the best of our knowledge, no static WCET estimation method so far has been proposed to estimate the impact of permanent faults of micro-architectural elements on worst-case execution times. Our work provides the first results on this topic, through the proposal of a static analysis method that evaluates the impact of faulty SRAM cells in instruction caches on WCETs. The trivial solution consisting in assuming all cache blocks as faulty is obviously safe. However, the probability that such a scenario occurs is very low (although non null) and such a method would largely overestimate the calculated WCET. Instead, our method, based on the probability that a SRAM cell is faulty (or will become faulty during the system lifetime), evaluates probabilistically how many additional cache misses (*fault-induced misses*) may occur. The distribution of the timing penalties due to faults is then used to derive probabilistic worst-case execution times. These probabilistic execution times can finally 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 commercial aerospace industry (Slijepcevic et al. 2013).

An essential benefit of our approach is that its probabilistic nature stems only from the probability associated with the presence of faults. By construction, the worst-case execution path cannot be missed, since it is determined using static analysis, extended to cope with permanent faults. This allows our method to be used in safety-critical real-time systems. In addition, as demonstrated by experimental results, the proposed method is computationally tractable, since it avoids the exhaustive enumeration of all possible fault locations. Experimental results also show that the proposed method accurately estimates WCETs in the presence of permanent faults compared to a method that explores all possible locations for faults.

On the one hand, the proposed method allows to quantify the impact of permanent faults on WCET estimates for chips with a known probability of cell failure for the whole chip lifetime. On the other hand, and most importantly, our work can also be used in architectural exploration frameworks to select the most appropriate fault management mechanisms, for current and future chip designs.

The remainder of the paper is organized as follows. Section 2 first surveys related work. The fault model and the base WCET estimation technique extended to cope with faults are presented in Sect. 3. Section 4 describes the proposed probabilistic WCET estimation method. Experimental results are presented in Sect. 5. Finally, we summarize the contributions of our research and give directions for future work in Sect. 6.

2 Related work

WCET estimation techniques may use *static analysis* or *measurements* to derive WCET bounds (Wilhelm et al. (2008). Our method falls into the first category, which by construction provides safe WCET bounds. 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 produces *probabilistic* WCET bounds to reflect the uncertainty of the location of faults in the architecture. It can be classified as *static probabilistic timing analysis* (SPTA) method according to the terminology introduced in the Proartis project (Cazorla et al. 2013).

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 last two decades (Theiling et al. 2000; Mueller 2000; Ferdinand and Wilhelm 1998; Hardy and Puaut 2008). These methods are by construction safe, provided that the details of the cache architecture are known. They are accurate if the cache structure, in particular the cache replacement policy, is carefully selected (Reineke et al. 2007). To the best of our knowledge, all static cache analysis methods assume fault-free caches, and are not sound anymore if the hardware is subject to permanent faults; our work is the first to integrate the effect of permanent faults in a static cache analysis method.

Some scheduling approaches exist to cope with faults while keeping time predictability. Most of them support *transient* faults using mechanisms such as checkpointing and rollback, or active replication. Such approaches aim at integrating the cost of error detection and error recovery/masking in the task schedulability analysis (Ghosh et al. 1997; Chevochot and Puaut 1999; Punnekkat et al. 2001). In the scope of real-time systems, comparatively little research has addressed the effect of permanent faults and disabling of resources on application performance.

The study presented in Hardy et al. (2012) addresses this issue for non real-time applications. They assume the same fault model as in our method, but concentrate on average-case performance, whereas our focus is on real-time applications and *worst-case* performance.

A method to statically compute probabilistic WCETs in the presence of faulty sensors is presented in Höfig (2012). Each configuration of faulty sensors yields a different path in the program, for instance, due to the interpolation of sensor values, and thus a different WCET. Probabilities of faulty sensors are derived using a safety analysis model and are then used to obtain the overall probabilistic WCET. In contrast to our work, (Höfig 2012) focusses on faulty sensors, whereas we focus on faulty SRAM cells in instruction caches.

The objective and assumptions in Slijepcevic et al. (2013) are very close to ours. A very similar fault model is used, and the objective 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, (Slijepcevic et al. 2013) uses a *measurement-based probabilistic timing analysis* method (MBPTA), that determines probabilistic WCETs based on measurements, whereas we use static probabilistic timing analysis (SPTA).

By construction MBPTA, in contrast to SPTA, does not necessarily explore all application paths, which can cause the longest path to be missed and thus does not provide an absolute guarantee that the computed probabilistic WCET is safe. Regarding the processor architecture, (Slijepcevic et al. 2013) assumes a PTA-compliant architecture (e.g. the architecture described in Kosmidis et al. 2013) that implements random cache placement and replacement). This is not a requirement in our method. In summary, the fundamental difference between the two approaches is that in Slijepcevic et al. (2013) the probabilistic nature of the estimated WCETs stems from several factors: uncertainty in the path exploration procedure, probabilistic hardware timing, probabilistic fault location and fault count. In contrast, in our method, the only source of uncertainty comes from the probability associated to faults.

3 Background

3.1 Architecture and fault model

The proposed 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. The architecture is considered free from timing anomalies,[1] meaning that a cache miss leads to the worst-case behavior. For the scope of the paper, a single level of instruction cache is assumed, and the data cache is assumed perfect.

Only *permanent* faults affecting the instruction cache are considered. The rest of the architecture is assumed to be fault free. The interaction between faulty micro-architectural components is considered out of the scope of this paper and left for future work. The effects of transient faults (e.g. particle strikes) are not addressed and are assumed to be covered by other error detection/correction codes (ECC).

In the cache, 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 by using hardening and/or redundancy techniques.

Faulty cache blocks are assumed to be detected using post-manufacturing and boot-time tests, ECC, and built-in self-tests. Cache block disabling has already been successfully used in commercial processors (McNairy and Mayfield 2005).

The term *faulty cache configuration* is used to denote a cache with faulty blocks. Such a configuration is represented by a vector with an entry per set, indicating its number of faulty blocks $\in [0, W]$. The exact position of the faulty blocks in each set has no importance, thanks to LRU replacement: in case of block failure, the LRU stack of a set is reduced by its number of faulty blocks.

Each SRAM cell (i.e. bit) is assumed to have an equal probability of failure (*pfail*). *pfail* corresponds to the probability of a bit to be permanently faulty due to process defects, aging, low-voltage operation, et cetera. Moreover, the locations of perma-

[1] Especially, an in-order pipeline with a constant latency per instruction is assumed like in Slijepcevic et al. (2013).

nently faulty SRAM cells are considered as random. This allows to account for some major causes of uncorrelated faults (Cheng et al. 2011).

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

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

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

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

3.2 Base static WCET estimation technique

The proposed analysis extends the standard static WCET estimation technique briefly described hereafter, originally designed for fault-free architectures. Since our focus is on the impact of permanent faults on instruction caches, the low-level analysis for the scope of this paper only includes static instruction cache analysis. The base static WCET estimation technique is representative of the current state-of-the-art and is implemented in several tools (Wilhelm et al. 2008). The reader is referred to (Wilhelm et al. 2008) and (Theiling et al. 2000) for further details on the static analysis techniques extended in our work.

3.2.1 Low-level analysis

The contribution of instruction caches to the WCET is determined by associating a Cache Hit/Miss Classification (CHMC) to every memory reference. The CHMC we use, defined in Theiling et al. (2000), represents the worst-case behavior of each reference with respect to the instruction cache:

- *always-hit* (AH): the reference will always result in a cache hit,
- *first-miss* (FM): the reference could neither be classified as hit nor as miss the first time it occurs but will result in cache hits afterwards,
- *always-miss* (AM): the reference will always result in a cache miss,
- *not-classified* (NC): the reference could neither be classified as hit nor as miss.

CHMCs are obtained by applying the static analysis technique described in Theiling et al. (2000), based on abstract interpretation. When using this technique, 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*,
- the *May* analysis determines if 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*.

In order to avoid enumerating all possible (concrete) cache contents, each of these three analyses compute 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. For instance, a block in the *Must* ACS at a given point is guaranteed to be in the cache at that point. Furthermore, for the *Must* and *Persistence* analyses, the age of a block in an ACS is the maximum possible age that block could have in the cache at run-time. The classifications of references as AH/FM are based on the maximal age of the reference in the corresponding ACS (e.g. a reference is classified AH if its age is lower or equal to w).

This concept of age in abstract cache states is at the core of the method. An in-depth understanding of static cache analysis methods is not mandatory to understand the method developed in this paper; further details can be found in Theiling et al. (2000).

3.2.2 High-level analysis

WCET calculation in this study 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 (Li and Malik 1995; Puschner and Schedl 1997). 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). An upper bound of the program's WCET is obtained by maximizing the following objective function:

$$\sum_{b \in BasicBlocks} T_b * x_b \tag{3}$$

T_b (constant in the ILP problem) is the timing information of basic block b. T_b integrates the effects of caches, and is computed using the CHMC of its references and the cache and memory latencies. x_b (variables in the ILP system, to be instantiated by the ILP solver) correspond to the number of times basic block b is executed.[2] The values of variables x_b, once set by the ILP solver to maximize the objective function, identify one of the paths in the program leading to the estimated WCET. The term WCEP will be used to denote one on the longest execution paths.

4 Probabilistic static WCET estimation for faulty caches

4.1 Design rationale

Cache blocks with permanent faults are disabled, which reduces the cache capacity and leads to additional cache misses, named all along the paper *fault-induced misses*;

[2] Due to the definition of CHMC FM, variable x_b is split into two distinct variables, one for the first execution of the block (x_b^{first}) and another for the next executions (x_b^{next}).

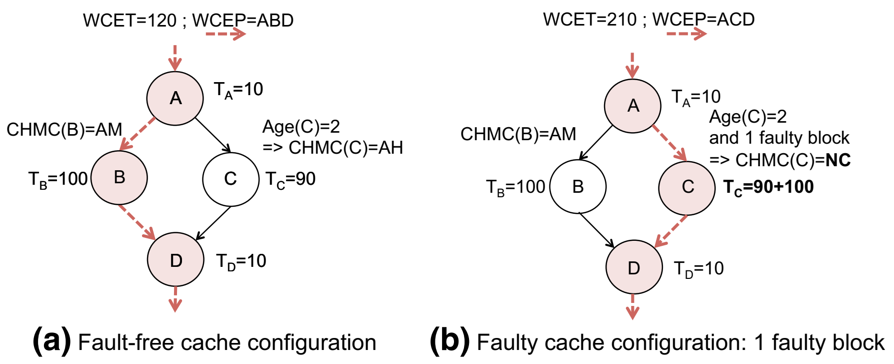


Fig. 1 WCET and WCEP estimation for an architecture with a fully-associative 2-way cache (1 cycle cache/100 cycles memory latency). A reference is performed by each basic block of the conditional structure. A WCEP variation occurs between the fault-free configuration (**a**) and the faulty cache configuration (**b**)

a reference that was known to hit the cache for a fault-free cache may result in a cache miss in the presence of faults.

Consequently, when statically analyzing faulty caches, some references that were classified as hits when ignoring faults may have to be changed to misses in the presence of faults. For instance, let us assume a 2-way set-associative cache. For a given reference, if the cache line containing that reference has a maximal age determined by the *Must* analysis equal to 2, the fault-free analysis will classify the reference as AH. However, if there is one faulty block in the set, the reference has to be classified as NC. The estimated WCET is thus affected by faults due to the possible modifications of the timing information of individual basic blocks.

The WCEP may change as well as compared to the fault-free WCEP. This behavior is illustrated by Fig. 1. In a fault-free execution (Fig 1a), basic block C is not on the WCEP because the memory reference in C is known to always hit the cache. With the faulty cache configuration (Fig 1b), if the cache block of C is faulty, this access is classified as a miss and changes the timing information of C, which now becomes part of the WCEP.

Due to this possible WCEP variation, there is a need to cover all paths when analyzing the impact of faults. To do so, a safe naive brute-force solution to compute the WCET probability distribution would consist in estimating for each possible faulty cache configuration the WCET and the corresponding probability of that configuration to happen.

The probability of a faulty cache configuration to occur would then be computed as follows:

$$\prod_{s=1}^{S} p_{wf}(nbfaulty(s)), \tag{4}$$

where the function *nbfaulty(s)* returns the number of faulty blocks of set s in the configuration.

However, this exhaustive computation is impractical, due to the prohibitive number of faulty cache configurations for which a WCET estimation has to be performed:

	MRU	LRU	Faulty cache configuration
SET 0	120	10	1
SET 1	150	14	1
SET 2	180	13	0
SET 3	220	20	0

Fig. 2 Number of references detected as hits per set and way, determined with the fault-free WCET computation for single-path code and a 2-way 4-set LRU cache. The number of fault-induced misses for the considered faulty cache configuration is equal to 10 + 14

$(W + 1)^S$. The method we propose to deal with faulty caches provides probabilistic WCET estimates very close to the one of the exhaustive method, but is computationally tractable. This brute-force method will only be used as a baseline for the validation of our approach in Sect. 5.

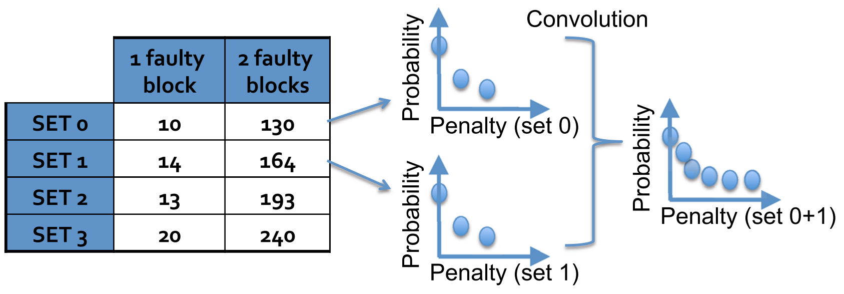
4.2 Method description

The proposed approach computes, for a given program, cache configuration, and probability of cell failure, a WCET probability distribution. In order to avoid an exhaustive computation of WCETs for all combinations of fault locations, the proposed approach operates in two steps. It first computes the fault-free WCET. 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 fault-free WCET estimation is performed using the static analysis described in Sect. 3.2. Thus, at the core of the method is the computation of the penalty probability distribution described hereafter. We first present how to compute the fault-induced miss penalty distribution in Sect. 4.2.1 and, then we propose a solution to reduce the pessimism of our approach in Sect. 4.2.2.

4.2.1 Computation of fault-induced miss penalty probability distribution

To progressively go into the details of our method, we first explain how the fault-induced miss penalty is computed from a known faulty cache configuration. We first focus on single-path code and then present the method on multi-path programs.

For single-path code, the number of fault-induced misses can be easily derived from the results of the fault-free WCET analysis, since the number of cache hits for each set on the (unique) execution path is known. This is depicted in Fig. 2. The figure shows for every set and every way, from the most recently used (MRU) to the LRU position in the LRU stack, how many accesses are detected as hits in that position. The position in the LRU stack is given by the maximal age of each reference provided by the cache analysis (*Must* and *Persistence* analyses). For instance, number 10 in the LRU position in set 0 means that 10 references are detected as hits in that position when set 0 is fault free. Thus, there are 10 + 14 fault-induced misses resulting from two faulty blocks, located in sets 0 and 1.



	1 faulty block	2 faulty blocks
SET 0	10	130
SET 1	14	164
SET 2	13	193
SET 3	20	240

(a) Fault miss map (FMM) **(b)** Computation of penalty probability distribution

Fig. 3 **a** Fault miss map (FMM) derived from the hits per sets and ways of Fig. 2. **b** Example of computation of the distribution of fault-induced miss penalties for the first two sets

However, due to the WCEP variation effect presented in Fig. 1, generalizing the method to multiple-path programs is not straightforward. Using the number of hits occurring along the fault-free WCEP may indeed lead to an underestimation of the number of references that cause misses and thus may lead to an underestimation of the fault-induced miss penalty.

To overcome this issue and ensure that the number of fault-induced misses is never underestimated, we compute for each set an *upper bound* of the number of references detected as hits instead of computing the exact value. This is performed separately for each set of the cache and for each possible number of faulty cache blocks $f \in [1, W]$ in the set. The computation is made insensitive to path variations by considering the memory references on *all* paths, and not only the ones on the WCEP. More precisely, for a given set s_j and a given number of faulty blocks f, the upper bound of the number of fault-induced misses is obtained by an ILP formulation where the objective function to maximize is as follows:

$$\sum_{b \in BasicBlocks} R_b * x_b \tag{5}$$

in which R_b stands for the number of references of basic block b which map onto set s_j and determined by the cache analysis to have a maximal age in the f latest LRU position. x_b is the number of times basic block b is executed. This ILP formulation is subject to the same set of linear constraints (structural constraints and loop bound constraints) as the IPET formulation used to compute the WCET (Sect. 3.2.2).

By solving this ILP system for all sets and all possible numbers of faulty blocks per set ($S * W$ different resolutions), we obtain a Fault Miss Map, denoted by FMM in the following. This map gives an upper bound of the number of fault-induced misses per set and per number of faulty blocks. In the map, a row corresponds to a set and each column corresponds to a number of faulty blocks. Figure 3a gives the FMM derived from the example of Fig. 2. For set 0, 10 fault-induced misses are determined when 1 block is faulty, which corresponds to the number of hits in the LRU position. 130 fault-induced misses are computed when the 2 blocks are faulty, which corresponds to the sum of the number of hits in the MRU and LRU position.

Given a faulty cache configuration and the FMM, the corresponding fault-induced latency is then:

$$Lat_{mem} * \sum_{s=0}^{S-1} \begin{cases} FMM[s][nbfaulty(s)] & \text{if } nbfaulty(s) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

with Lat_{mem} representing the memory latency.

In order to avoid exploring all possible faulty cache configurations to obtain the penalty probability distribution, our method takes benefit of the *independence of sets*. The penalty distribution is first computed for each cache set independently by studying the respective impact of $f \in [1, W]$ faults. For each set, 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} (Eq. 2). Since sets are independent and all possible memory accesses are considered, all these discrete distributions are independent. Thus, they can be combined by computing the product of their probability distributions (also called convolution) to obtain an upper bound of the penalty distribution, as illustrated in Fig. 3b. In the figure, the penalty probability distribution of set 0 and set 1 have 3 distinct points corresponding to 0, 1 and 2 blocks faulty 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.

To improve the solver's computation time of the different ILP resolutions, we inject fault-insensitive frequency values, obtained during the fault-free WCET estimation, into the ILP system before its resolution. This is achieved by keeping track of each x_b value determined during the fault-free WCET estimation and by using a dominance analysis to determine basic blocks which are always on the execution path (i.e. they dominate all the exit points of the program). The maximal number of times a basic block b is executed is fault-insensitive if basic block b is always on the execution path. For such a basic block, the corresponding variable x_b is set to the value obtained during the fault-free WCET estimation before the resolution.

4.2.2 Tightness improvement

While the method as described so far never underestimates the number of memory references that may be affected by faults, it may be pessimistic. Indeed, mutually exclusive accesses (i.e. accesses performed along the two branches of an if-then-else construct) mapped to different sets are *all* considered as causing fault-induced misses. This phenomenon is particularly important when entire sets are faulty, because references target the MRU position more significantly than the other positions. To improve the tightness of the distribution of penalties at a reasonable computation cost, we focus on such situations.

We estimate the worst-case penalty when sf sets ($sf \in [1, S]$) are entirely faulty and $S - sf$ sets have all their blocks faulty except the MRU block. Let us denote by $penalty_{sf}$ the computed worst-case penalty. After having computed the distribution of penalties as described before (Sect. 4.2.1), the distribution is corrected to eliminate overestimated penalties. For each value of the distribution corresponding to *sf* entirely faulty sets, none of the corresponding penalty values can exceed $penalty_{sf}$ meaning that the values above that penalty can safely be reduced to $penalty_{sf}$.

To estimate the worst-case penalty for *sf* entirely faulty sets, the IPET formulation presented in Sect. 3.2.2 is extended to compute the WCET in the presence of *sf* entirely faulty sets. The worst-case penalty is then obtained by subtracting the fault-free WCET already computed.

In the ILP formulation, the presence of an entirely faulty set s is represented by a binary variable sf^s set to 1 when all blocks in set s are faulty. The following constraint limits the number of sets that can be entirely faulty to *sf*:

$$\sum_{s=0}^{S-1} sf^s \leq sf \tag{7}$$

The penalty can be separated in two distinct parts: (i) the penalty resulting from all accesses that hit exclusively in the MRU position (denoted by P_b^s for the accesses of basic block b that hit in the MRU position of set s) and, (ii) the penalty resulting from all accesses that hit but not in the MRU position (denoted by P_b' for the accesses of basic block b). Thus, the IPET objective function can be extended as follows:

$$\sum_{b \in BasicBlocks} \left(T_b + P_b' + \sum_{s=0}^{S-1} P_b^j * sf^s \right) * x_b \tag{8}$$

In this formula, the penalty due to accesses in the MRU position is represented by the sum of the penalties coming from the MRU position of each set s times sf^s.

However, Eq. 8 is in a quadratic form, because sf^s and x_b are two variables of the ILP system. To obtain a linear formulation, we introduce the variable h_b^s which is equal to x_b if sf^s is set to 1 and equal to 0 otherwise, which is modeled by the following constraints:

$$h_b^s \leq x_b \quad \text{and} \quad h_b^s \leq M * sf^s, \tag{9}$$

where M is a constant higher than any possible x_b. In our experiments M is fixed to MAX_INT.

The objective function can thus be rewritten as follows:

$$\sum_{b \in BasicBlocks} (T_b + P_b') * x_b + \sum_{s=0}^{S-1} P_b^s * h_b^s \tag{10}$$

Finally, to reduce the explored solution space and thus to improve the solver's resolution time, an extra constraint on the objective function is added, indicating an upper bound of that function. This upper bound corresponds to the fault-free WCET plus the extra-latency stemming from faults, the latter being the sum of all the fault-induced cache misses, multiplied by the memory latency. Formally:

$$(10) \leq WCET_{fault\,free} + \sum_{s=0}^{S-1} FMM[s][W] * Lat_{mem} \qquad (11)$$

S distinct resolution of the above ILP formulation are needed to determine the values of $penalty_{sf}$.

4.3 Complexity considerations

The base method that was presented in this section requires one run of the WCET estimation tool to estimate the fault-free WCET. Then, $S * W$ resolutions of ILP systems are required to obtain the fault miss map (FMM). Finally, to obtain the distribution of fault-induced miss penalties, convolutions of the distributions of the S sets have to be performed (complexity $O(log(S))$). This complexity has to be compared to the $(W + 1)^S$ runs of the WCET estimation tool that would be required if all faulty configurations were exhaustively examined. The method proposed to improve the tightness requires S additional ILP system resolutions. Measured analysis times will be given in Sect. 5.

5 Experimental results

In this section, we first present the experimental setup. Then the experiments demonstrating the accuracy as well as the applicability of the method are detailed.

5.1 Experimental setup

Analyzed codes The experiments were conducted on 25 benchmarks from the Mälardalen WCET benchmark suite.[3] Table 1 summarizes the applications' characteristics. The analyzed programs include both single-path programs (*fdct, jfdctint, mincer, prime, matmult*) and multi-path programs. They include both small loop intensive programs and control programs, containing fewer loops and more complex control flow.

Cache configurations The instruction cache size is fixed to 1 KB (or it will be mentioned otherwise) and an LRU replacement policy is assumed. The cache and memory latencies are fixed to 1 cycle and 100 cycles respectively. The different configurations in terms of associativity and cache line size are given when describing every experiment. A cache block is composed of the data bytes, the tag bits and their corresponding ECC bits assumed to be SEC-DED codes (Hamming 1950). As an example, for 1 KB 4-way cache with 64B cache line, for each cache block, there are 23 bits for the tag and 6 bits for the tag ECC, as well as 11 ECC bits to protect the data.

WCET analysis The experiments were conducted on MIPS R2000/R3000 binary code compiled with gcc 4.1 with no optimization and the default linker memory layout. The WCETs of tasks are computed by the Heptane static WCET estimation tool (Colin

[3] http://www.mrtc.mdh.se/projects/wcet/benchmarks.html.

Table 1 Analyzed benchmarks

Name	Description	Binary code size (bytes)
adpcm	Adaptive pulse code modulation algorithm	7,644
bs	Binary search for the array of 15 integer elements	328
bsort100	Bubblesort program	588
cnt	Counts non-negative numbers in a matrix	816
compress	Data compression program (adopted from SPEC95 for WCET-calculation)	2,724
crc	Cyclic redundancy check computation on 40 bytes of data	1,368
expint	Series expansion for computing an exponential integral function	940
fdct	Fast Discrete cosine transform	3,368
fft	1,024-point fast Fourier transform using the Cooly–Turkey algorithm	6,244
fir	Finite impulse response filter (signal processing algorithms) over a 700 items long sample	4,460
insertsort	Insertion sort on a reversed array of size 10	472
jfdctint	Discrete-cosine transformation on a 8×8 pixel block	3,048
lcdnum	Read ten values, output half to LCD	608
ludcmp	LU decomposition algorithm	3,092
matmult	Matrix multiplication of two 20×20 matrices	848
minver	Inversion of floating point matrix	5,312
ndes	Complex embedded code	3,820
ns	Search in a multi-dimensional array	600
nsichneu	Simulate an extended Petri net	44,028
prime	Calculates whether numbers are prime	512
qurt	Root computation of quadratic equations	2,412
select	A function to select the nth largest number in a floating point array	1,892
sqrt	Square root function implemented by Taylor series	780
statemate	Automatically generated code generated by the STAtechart real-time-code generator STARC	8,768
ud	Calculation of matrixes	2,348

and Puaut 2001), that implements the cache analysis and WCET computation steps as presented in Sect. 3. 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. CHMC *NC* is thus considered equivalent to *AM*. Cplex 12.5 is used to solve the different ILP systems.

5.2 Experimental results

In this section, the accuracy of the proposed methods is first evaluated. Measurements of the analysis computation time are then provided. Here, the optimization exploiting the dominance relation, as presented in Sect. 4.2.1 is systematically used. Finally, some first results on the impact of architectural parameters are given.

5.2.1 *Accuracy of probabilistic WCET estimates*

We evaluate the quality of the proposed method in two steps, presented in the next two paragraphs. First, we compare the base method (Sect. 4.2.1), named *Base* in the following, and tightness-improved method (Sect. 4.2.2), named *Improved* in the following, to a baseline method that enumerates all possible fault locations (named *Exhaustive* hereafter) and statically evaluates the WCET for each of them. This first set of experiments aims to show that the WCET produced by the proposed method is very close to the exhaustive method. Second, we compare estimated WCETs with a baseline that simulates program execution with random selection of fault location. The term *accuracy* is used in both situations to quantify the difference with the baseline.

Results will be presented by depicting the complementary cumulative distribution function. This curve can be seen has an exceedance function which indicates for a given targeted probability p (e.g. 10^{-15} per task activation for aerospace commercial industry Slijepcevic et al. 2013) the value at which the random variable WCET will be equal to, or below that value, with probability $1 - p$.

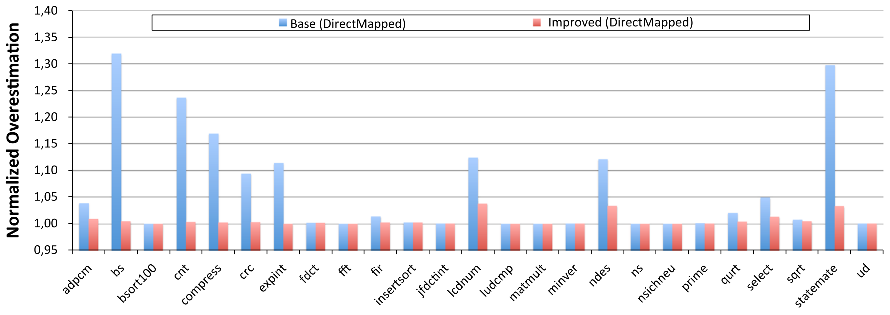
In this set of experiments, the instruction cache size is fixed to 1 KB with a 64B cache line. Two configurations are used with 1-way (named *DirectMapped*) and 2-way (named *SetAssoc*) per set respectively. These two configurations have a sufficiently small number of different faulty cache configurations ($6{,}561 = 3^8$ for *SetAssoc* and $65{,}536 = 2^{16}$ for *DirectMapped*) to allow an exhaustive computation/simulation of the WCET distribution for all possible fault locations.

The probability of failure *pfail* is fixed to 10^{-4} which is representative of the highest assumed probability of cell failure in related work (Nassif et al. 2010; Slijepcevic et al. 2013). Other smaller *pfails* have been used during our experiments (see Sect. 5.2.5) and we found that the worst accuracy occurred when the highest *pfail* was used.

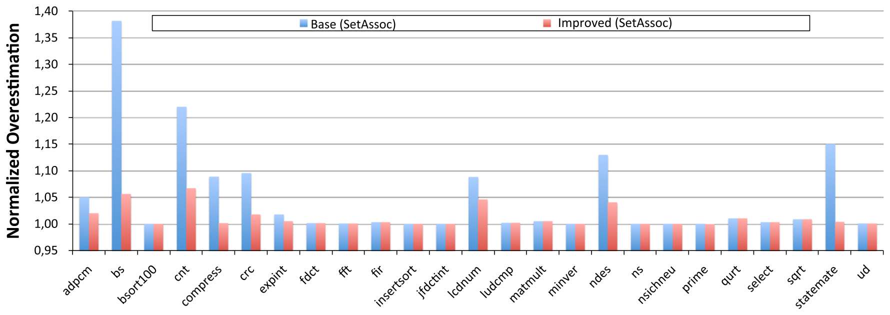
5.2.2 *Comparison of our method with static analysis and exhaustive exploration of fault locations*

To show the accuracy for all benchmarks, we first provide the normalized overestimation of the *Base* and *Improved* methods against the *Exhaustive* method when the targeted probability is set to 10^{-15}. The results are depicted in Fig. 4 for both cache configurations. The figure shows that method *Improved* significantly improves the accuracy of method *Base* for both cache configurations. Moreover, WCETs obtained by method *Improved* are very close to the ones obtained by method *Exhaustive*. A deeper analysis of the results, for all benchmarks, reveals that the benchmarks can be classified into three different categories according to the accuracy of our method:

- *perfect* (identical to the *Exhaustive* method): *fdct, jfdctint, minver, prime, ud, matmult, nsichneu, ns, insertsort, bsort100*
- *very accurate* (very close to the *Exhaustive* method): *adpcm, fir, qurt, sqrt, fft, ludcmp*



(a) DirectMapped cache configuration



(b) SetAssoc cache configuration

Fig. 4 Normalized overestimation of the base and improved methods against the exhaustive when the targeted probability is set to 10^{-15} $(pfail = 10^{-4})$

– *accurate* (close to the *Exhaustive* method): *bs, cnt, compress, crc, expint, lcdnum, ndes, select, statemate*

For space considerations, only a subset of the complementary cumulative distributions of the analyzed 25 benchmarks are given in Fig. 5 for the two considered cache configurations (*DirectMapped* on the left, *SetAssoc* on the right). Presented results are selected to have one benchmark of the three categories identified above. For the last two categories, the benchmarks are chosen to be representative of the worst observed accuracy. Each graph depicts the complementary cumulative distribution function of the *Exhaustive*, *Base* and *Improved* methods. The 10^{-15} probability is highlighted by an horizontal line in all graphs. The x-axis corresponds to the WCET (in cycles) and the y-axis, in log scale, corresponds to the probability. By construction, the *Base* and *Improved* curves are two upper bounds of the *Exhaustive* distribution meaning that they are equal to or above the *Exhaustive* curve.

The general observation that can be made on all graphs is that the presence of permanent faults significantly increases WCET estimates compared to the fault-free WCET.

Regarding accuracy, for the *perfect* category, represented by *nsichneu* in Fig. 5, the *Base* and *Improved* curves are identical to the *Exhaustive* distribution. This category is composed of two groups of benchmarks: single-path and multi-path applications.

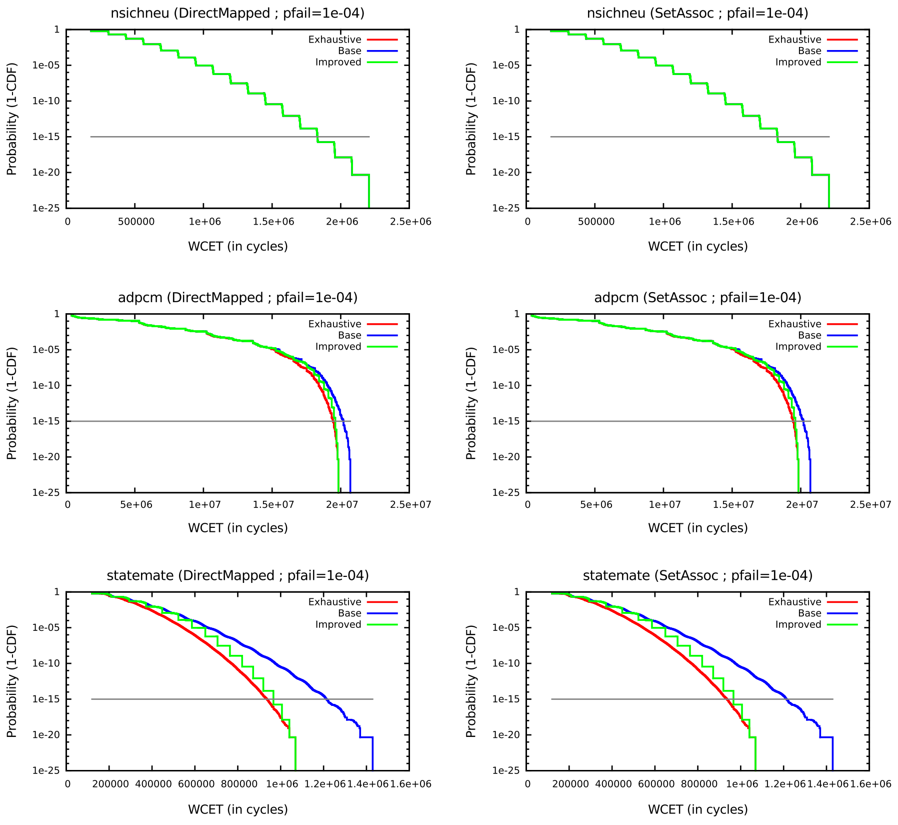


Fig. 5 Complementary cumulative distribution function for nsichneu, adpcm and statemate

For the former group, this behavior is expected since the memory accesses are known and are the same whatever the execution conditions. For the latter group, we found that these benchmarks (*nsichneu, ud, ns, insertsort, bsort100*) do not have *else* blocks in the conditional constructs of their code. This means that even if they are not single-path, there is no overestimation for these codes of the number of memory references when computing the impact of faults.

The second category, illustrated by *adpcm* in Fig. 5, reveals the presence of a small overestimation of the *Base* curve as compared to the *Exhaustive* curve, and this when nearly all the cache blocks are faulty (right part of the distribution). This overhead is due to *if-then-else* constructs in their code which induces a few additional memory references in the *Base* method as compared to the *Exhaustive* method. The *Improved* method eliminates most of this overestimation and the resulting distribution is very accurate.

The last category, depicted in Fig. 5 with *statemate*, shows that a significant overestimation can be observed (up to nearly 40 %) when comparing the *Base* method to the *Exhaustive* method. For this set of benchmarks, we found that their code contains *if-then-else* or *switch* constructs inside loops, which results in a significant overestimation of the memory references in the *Base* method. The *Improved* method reduces significantly this overestimation, and even in that case, the *Improved* method produces accurate distributions.

Finally, in all experiments, even if the shapes of the curves vary across the cache configurations, the accuracy of our method is only marginally affected by the cache configuration. The code structure is the main factor impacting the method accuracy.

5.2.3 Comparison of our method with simulation and exhaustive exploration of fault locations

The second baseline, named *Run Exhaustive* hereafter, uses simulation-based results. *Run Exhaustive* is obtained by using the Nachos educational operating system,[4] running on top of a simulated MIPS processor. We have extended Nachos to evaluate the impact on the WCET of all possible faulty cache configurations. This baseline is used to show the impact on the accuracy of the static WCET analysis used in our approach as compared to the actual WCET.

In this set of experiments, only single-path benchmarks are used to compare our *Improved* method using static analysis against simulations. Only single-path programs are studied here to focus on the impact of faults on the method accuracy, and rule out imprecisions stemming from the determination of the longest execution path.

The results of these experiments are given in Fig. 6 for the two considered cache configurations (*DirectMapped* on the left, *SetAssoc* on the right) on benchmark *jfdctint*. Benchmark *jfdctint* is chosen because it has the worst observed accuracy (all the others are nearly equal to the baseline curve).

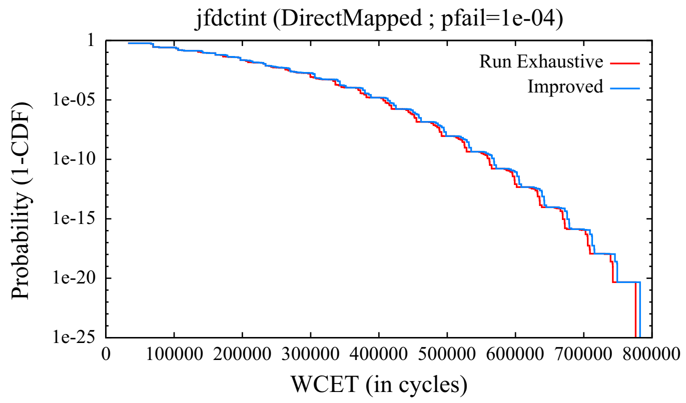
Even on the worst-performing benchmark, our approach is shown to be very accurate. The reason for the minor overestimation on benchmark *jfdctint* was investigated. We found that the WCET analysis slightly overestimate both the age of the references during the cache analysis and thus the number of memory accesses during the IPET analysis.

5.2.4 Computation time

The measurements provided below are performed using an Intel Core i7 (2.9 GHz) under OSX 10.8.2 with 8GB of DDR3 (1,600 MHz). Cplex 12.5 is set to run up to 4 threads in parallel to solve the ILP systems.

For the previous two cache configurations (1 KB cache, 64B lines, *DirectMapped* and *SetAssoc*), the longest benchmark to analyze is *nsichneu* which is the biggest benchmark used in our experiments (code size of 43 KB). For that benchmark, the computation time needed to get the fault-free WCET is 0.26 s and the time to compute the FMM is around 2 s for both cache configurations. The computation of the different worst-case penalties for that benchmark takes respectively 3 s for the *SetAssoc* con-

[4] Nachos web site, http://www.cs.washington.edu/homes/tom/nachos/.



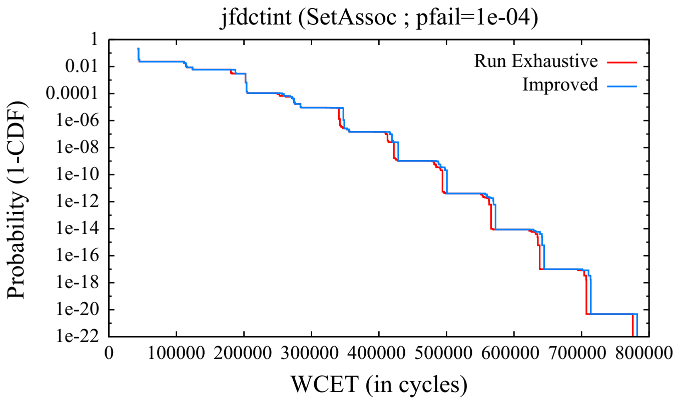


Fig. 6 Run exhaustive versus improved: complementary cumulative distribution function for jfdctint

figuration and 5 s for the *DirectMapped* configuration.[5] When looking at the other benchmarks, the computation takes at most 5 s, all steps included. To put it in perspective, when performing an exhaustive computation, more than 1,600 s (*SetAssoc*) and 16,000 s (*DirectMapped*) would be needed for *nsichneu*.

We have also considered bigger caches in this experiment to show the scalability of our method. The assumed cache structure is a 4-way associative cache with 64B cache lines and the number of sets vary from 4 to 256, for an overall cache size varying from 1 to 64 KB. The results for *nsichneu* are provided in Fig. 7 where the x axis represents the cache size and the y-axis in log scale represents the execution time in seconds. The *nsichneu* benchmark is selected because it produces the worst overall execution time in most of the cases.

For the largest cache size, the overall execution time needed is less than 800 s which shows the applicability of our approach on nowadays caches.

[5] In the conference version of the paper (Hardy and Puaut 2013) it was 270 s. The significant improvement comes from our complexity enhancement method consisting of injecting fault-insensitive frequency values into the different ILP systems (Sect. 4.2.1).

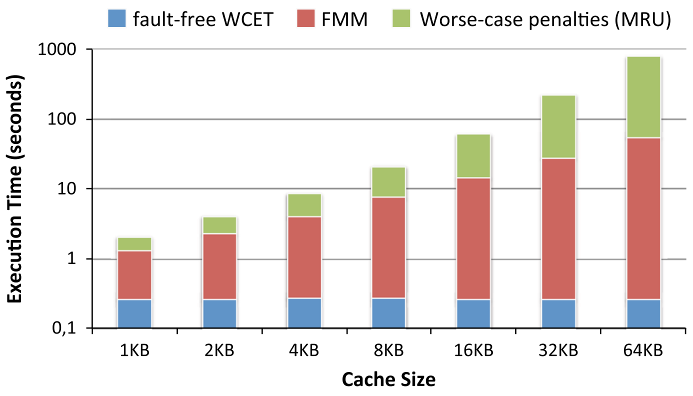


Fig. 7 Execution time measurement of a 1–64 KB (4-way 64B) cache

As expected, the fault-free WCET computation time stays constant because the cache size has no impact on the fault-free WCET ILP system. The FMM computation needs at most 56 s. A factor close to 2 is observed for the FMM computation when the cache size is doubled. Finally, the computation of the different worst-case penalties takes at most 742 s and the factor when the cache size is doubled is 3.4 in average.

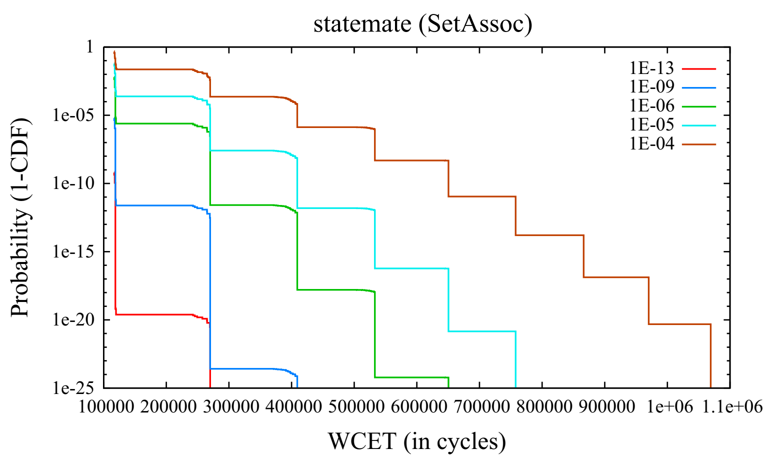
Concerning the convolution part, the number of discrete points becomes huge when the cache size is above 4 KB (at most $(W+1)^S = 5^{16}$ distinct points for a 4 KB cache 4-way 64B). Therefore, there is a need to use a re-sampling method (Maxim et al. 2012; Hardy et al. 2012) when performing the convolutions. However, this is not an issue since (Hardy et al. 2012) has shown that it takes around 30 s to perform a similar computation when considering a 2MB cache to get accurate results.

5.2.5 *Impact of pfail on WCET distributions*

As seen during the experiments evaluating the accuracy of our method, the impact of permanent faults on the WCET is significant when considering the highest *pfail*.

In this experiment, we explore the impact on WCET at different technology nodes of permanent faults. Figure 8 shows the results for the *SetAssoc* cache configuration (1 KB cache, 64B lines, associativity of 2) with the *Improved* method when considering a range of pfail from 10^{-13} to 10^{-4} representative of technology nodes from 45 to 12 nm (Nassif et al. 2010). The two selected benchmarks (*statemate* and *adpcm*) are representative of the two different observed behaviors.

At the current technology node (45 nm/pfail = 10^{-13}), we observe that the impact is very low for *adpcm* and almost null for *statemate* for the lowest targeted probability (10^{-15}). At 32 nm (*pfail* = 10^{-9}) or below, representative of future technology nodes, the impact becomes significant. Therefore, it is essential to mitigate the effect of such faults to minimize their resulting penalty. The proposed method facilitates the exploration of design tradeoffs to address this challenge.



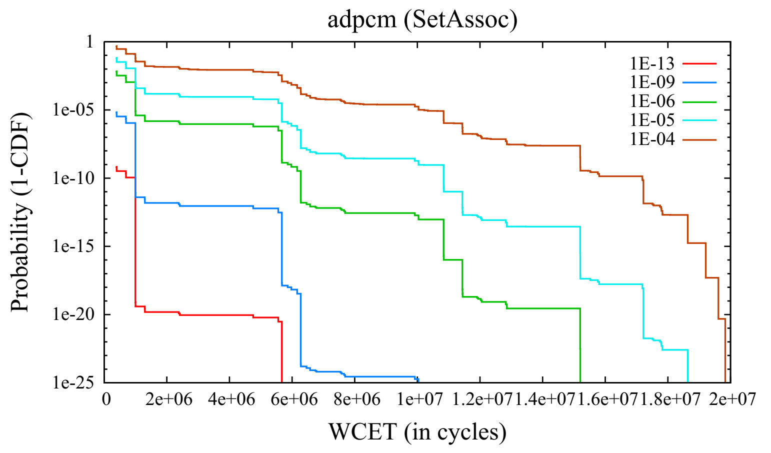


Fig. 8 Impact of permanent faults on WCET at different technology nodes (pfail)

5.2.6 Exploration of architectural parameters

In this section, we explore the cache parameters as a first step in the exploration of the design tradeoffs for supporting faulty caches for future technology nodes. In the following, the *Improved* method is used assuming $pfail = 10-4$ and a 1 KB cache where the associativity, the cache block size and the cell size vary.

We first present the impact of the cache associativity on the WCET distribution. The WCET distributions obtained by our method for three associativity degrees (1, 2 and 4) are presented in Fig. 9 for *jfdctint*. The three depicted cache configurations have a fixed cache line size of 64B and they only differ by their associativity degree.

The results show that the higher the associativity, the lower the impact on the WCET. A similar behavior is observed for all benchmarks. In other words, selecting a set-associative cache instead of a direct-mapped can significantly reduce the WCET that has to be considered for a given targeted probability. Having all blocks faulty in

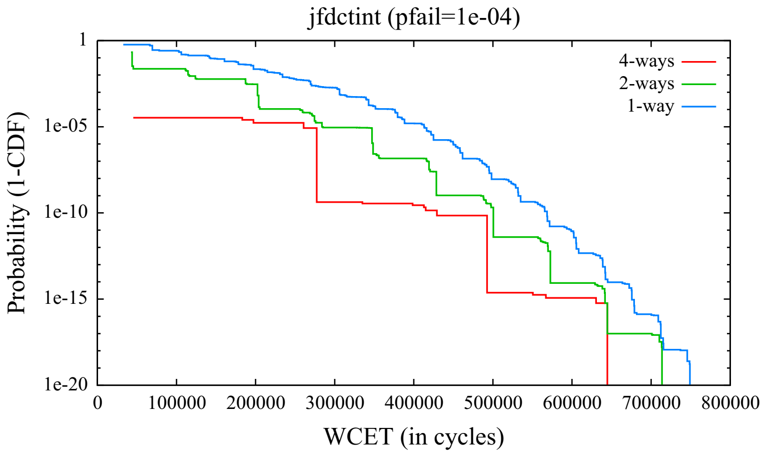


Fig. 9 Impact of the cache associativity on the WCET distribution

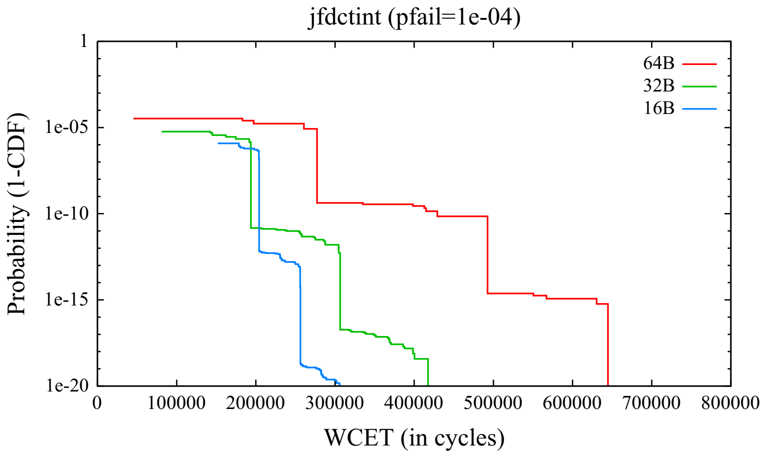


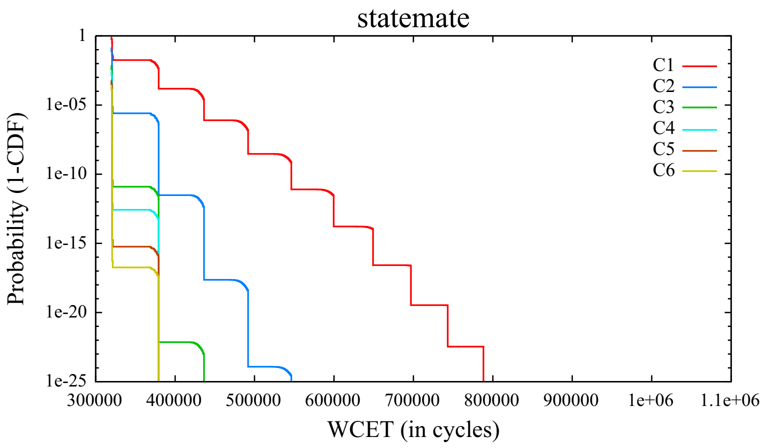
Fig. 10 Impact of the cache block size on the WCET distribution

a set has a strong influence on worst-case performance; the probability to have all the blocks faulty in a set for set-associative caches is lower than the probability to have one block faulty in a direct-mapped cache.

The second studied cache parameter is the size of the cache blocks. Based on the previous observation, we focus on 4-way set associative cache with respectively 16B, 32B and 64B cache block size. The WCET distributions are depicted in Fig. 10 for *jfdctint*. The results show that even if the fault-free WCET is higher when smaller cache blocks are used, the obtained distribution highlights the fact that smaller blocks are profitable when considering the impact of permanent faults. This conclusion can be made for all the benchmarks and is explained by the fact that the probability of a block failure depends on the size of blocks, the lower the number of bits in a block, the lower is the probability that the block is faulty.

Table 2 Relative cell area and *pfail* for different cell size (32 nm) (Zhou et al. 2010)

	C1	C2	C3	C4	C5	C6
Relative cell area	1	1.12	1.23	1.35	1.46	1.58
pfail	1.2e−03	1.2e−04	5.6e−06	2.1e−06	4.6e−07	1.9e−07



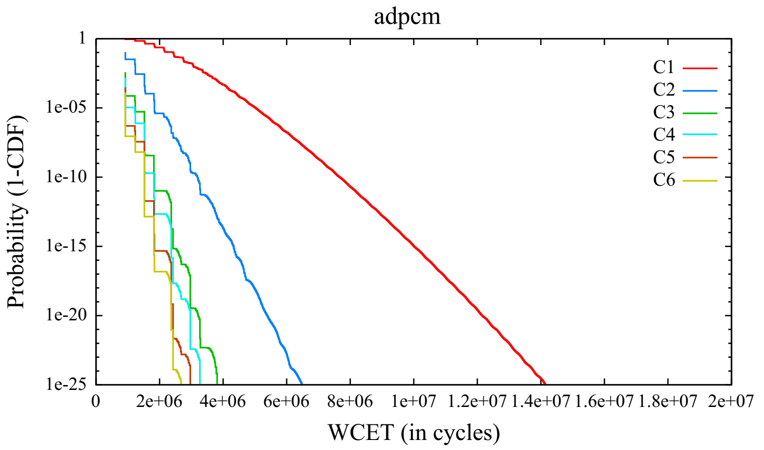


Fig. 11 Impact of permanent faults on WCET for different cell size

The last cache parameter studied in this experiment is the cell size. Using more robust cells can help reduce the impact of permanent faults on WCET. Based on previous observations, we focus on 4-way set associative cache with 16B cache line and to assess the benefits of cell size, we use six different cells sizes with their corresponding *pfail* from (Zhou et al. 2010) for 32 nm technology assuming a voltage of 0.7. The corresponding *pfail* and relative cell area are provided in Table 2.

Figure 11 shows the results for *statemate* and *adpcm* which are representative of the two different observed behaviors. An obvious observation is that the higher the cell size the lower the impact of faults on WCET because their *pfail* are lower. Using the most robust cells (C6) allows to significantly reduce the impact of such faults for *adpcm* and to be very close to the fault-free WCET for *statemate* when targeting a probability of 10^{-15}. However, this obviously comes at the cost of an increase in terms of die area. Thus, depending on the targeted probability, using the most robust cell may not be the best tradeoff since the same probability can be reached with lower cell size. For instance, when targeting a probability of 10^{-10} C3 is the best tradeoff between probability and die area for *statemate* under the selected experimental conditions.

To put it in a nutshell, choosing carefully the cache parameters at design time can help to limit the impact of faults on the WCET distribution.

6 Conclusions and future work

In this paper, we have proposed a method to calculate a probabilistic WCET bound in the presence of permanent faults in instruction caches. The method is based on static analysis, and as such is guaranteed to always find the longest execution paths. Its probabilistic nature only comes from the presence of faulty blocks in the architecture. The method is shown to be computationally tractable, and does not require an exploration of all the possible locations of faults. Experimental results show that the method is accurate, in the sense that its provides probabilistic WCETs close to (but never under) the method that exhaustively explores all possible locations for faults.

So far, our method has focused on faults in instruction caches. Generalizing it to other micro-architecture components with SRAM cells (e.g. branch predictors, data caches, cache hierarchies) and studying the interactions between architectural components is a first direction for future work. Comparison with similar methods based on measurements (Slijepcevic et al. 2013) is another direction for future work. A more general direction would be to integrate our method in an architectural exploration framework, to decide of the most appropriate fault management mechanism for all architectural elements.

Acknowledgments The authors would like to thank Jaume Abella, Benjamin Lesage, Bastien Pasdeloup, Erven Rohou and André Seznec for their fruitful comments on earlier versions of this paper.

References

Borkar S, Karnik T, Narendra S, Tschanz J, Keshavarzi A, De V (2003) Parameter variations and impact on circuits and microarchitecture. In DAC40, pp 338–342

Bowman K, Tschanz J, Wilkerson C, Lu SL, Karnik T, De V, Borkar S. (2009) Circuit techniques for dynamic variation tolerance. In: DAC46. ACM, New York, pp 4–7. doi:10.1145/1629911.1629915

Cazorla FJ, Quiñones E, Vardanega T, Cucu L, Triquet B, Bernat G, Berger E, Abella J, Wartel F, Houston M, Santinellei L, Kosmidis L, Lo C, Maxim D (2013) Proartis: probabilistically analysable real-time systems. ACM Trans Embed Comput Syst 12:79

Cheng L, Gupta P, Spanos CJ, Qian K, He L (2011) Physically justifiable die-level modeling of spatial variation in view of systematic across wafer variability. IEEE Trans CAD Integr Circuits Syst 30(3):388–401

Chevochot P, Puaut I (1999) Scheduling fault-tolerant distributed hard real-time tasks independently of the replication strategies. In: 6th international conference on real-time computing and applications symposium, pp 356–363

Colin A, Puaut I (2001) A modular and retargetable framework for tree-based WCET analysis. In: Euromicro conference on real-time systems (ECRTS), Delft, pp. 37–44

Engblom J (2002) Processor pipelines and static worst-case execution time analysis. PhD thesis, Uppsala University.

Ferdinand C, Wilhelm R (1998) On predicting data cache behavior for real-time systems. In LCTES '98: proceedings of the ACM SIGPLAN workshop on languages, compilers, and tools for embedded systems, pp 16–30

Ghosh S, Melhem R, Mossé D (1997) Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems. IEEE Trans Parallel Distrib Syst 8(3):272–284

Hamming R (1950) Error detecting and error correcting codes. Bell Syst Tech J 26(2):147–160

Hardy D, Puaut I (2008) WCET analysis of multi-level non-inclusive set-associative instruction caches. In: Proceedings of the 29th real-time systems symposium, pp 456–466

Hardy D, Puaut I (2013) Static probabilistic worst case execution time estimation for architectures with faulty instruction caches. In: 21st international conference on real-time networks and systems, RTNS 2013, Sophia Antipolis, October 17–18, pp 35–44

Hardy D, Sideris I, Ladas N, Sazeides Y (2012) The performance vulnerability of architectural and non-architectural arrays to permanent faults. In: Proceedings of the 45th annual IEEE/ACM international symposium on microarchitecture, MICRO'12, pp 48–59

Höfig K (2012) Failure-dependent timing analysis: a new methodology for probabilistic worst-case execution time analysis. In: Schmitt J (ed) Measurement, modelling, and evaluation of computing systems and dependability and fault tolerance. Lecture notes in computer science, vol 7201. Springer, Berlin, pp 61–75

Kosmidis L, Abella J, Quiñones E, Cazorla FJ (2013) A cache design for probabilistically analysable real-time systems. In: Proceedings of the conference on design, automation and test in Europe, DATE '13, pp 513–518

Li X, Roychoudhury A, Mitra T (2006) Modeling out-of-order processors for WCET estimation. Real Time Syst J 34(3):195–227

Li YTS, Malik S (1995) Performance analysis of embedded software using implicit path enumeration. In: Gerber R, Marlowe T (eds) LCTES '95: proceedings of the ACM SIGPLAN 1995 workshop on languages, compilers, & tools for real-time systems, vol 30, New York, pp 88–98

Maxim D, Houston M, Santinelli L, Bernat G, Davis RI, Cucu-Grosjean L (2012) Re-sampling for statistical timing analysis of real-time systems. In: Proceedings of the 20th international conference on real-time and network systems, RTNS '12. ACM, New York, pp. 111–120. doi:10.1145/2392987.2393001

McNairy C, Mayfield, J (2005) Montecito error protection and mitigation. In: HPCRI '05: 1st workshop on high performance computing reliability issues

Mueller F (2000) Timing analysis for instruction caches. Real Time Syst 18(2–3):217–247

Nassif SR, Mehta N, Cao Y (2010) A resilience roadmap. In: DATE, pp 1011–1016

Punnekkat S, Burns A, Davis R (2001) Analysis of checkpointing for real-time systems. Real Time Syst 20(1):83–102

Puschner P, Schedl AV (1997) Computing maximum task execution times: a graph based approach. Proc IEEE Real Time Syst Symp 13:67–91

Reineke J, Grund D, Berg C, Wilhelm R (2007) Timing predictability of cache replacement policies. Real Time Syst 37(2):99–122

Slijepcevic M, Kosmidis L, Abella J, Quinones E, Cazorla F (2013) DTM: degraded test mode for fault-aware probabilistic timing analysis. In: 2013 25th Euromicro conference on real-time systems (ECRTS), pp 237–248

Theiling H, Ferdinand C, Wilhelm R (2000) Fast and precise WCET prediction by separated cache and path analyses. Real Time Syst 18(2–3):157–179

Wilhelm R, Engblom J, Ermedahl A, Holsti N, Thesing S, Whalley D, Bernat G, Ferdinand C, Heckmann R, Mitra T, Mueller F, Puaut I, Puschner P, Staschulat J, Stenström P (2008) The worst-case execution-time problem-overview of methods and survey of tools. ACM Trans Embed Comput Syst 7(3):36:1–36:53. doi:10.1145/1347375.1347389

Zhou ST, Katariya S, Ghasemi H, Draper S, Kim NS (2010) Minimizing total area of low-voltage SRAM arrays through joint optimization of cell size, redundancy, and ECC. In: 2010 IEEE international conference on computer design (ICCD), pp 112–117. doi:10.1109/ICCD.2010.5647605