Random Modulo: a New Processor Cache Design for Real-Time Critical Systems

Carles Hernandez[‡], Jaume Abella[†], Andrea Gianarro[‡], Jan Andersson[‡], Francisco J. Cazorla[†,⋆]
[†]Barcelona Supercomputing Center (BSC-CNS), Barcelona (Spain)
[‡]Cobham Gaisler, Gothenburg (Sweden)
[⋆]Spanish National Research Council (IIIA-CSIC), Barcelona (Spain)

ABSTRACT

Cache memories have a huge impact on software's worst-case execution time (WCET). While enabling the seamless use of caches is key to provide the increasing levels of (guaranteed) performance required by automotive software, caches complicate timing analysis. In the context of Measurement-Based Probabilistic Timing Analysis (MBPTA) – a promising technique to ease timing analyis of complex hardware – we propose Random Modulo (RM), a new cache design that provides the probabilistic behavior required by MBPTA and with the following advantages over existing MBPTA-compliant cache designs: (i) an outstanding reduction in WCET estimates, (ii) lower latency and area overhead, and (iii) competitive average performance w.r.t conventional caches.

1. INTRODUCTION

The complexity of on-board computing systems in cars has steadily increased in recent years due to the innovation in their electronics and software components. On-board software in the automotive industry already comprises more than 100 millions lines of code [8] and it is expected to further increase in the foreseeable future. To execute the required software functionalities timely, more complex hardware comprising high-performance features is required.

Timing verification of automotive software is a critical step of software testing process. Worst-Case Execution Time (WCET) estimates should be sound and tight, but this is challenged by complex hardware. In particular, cache memories, which are ubiquitous in processor designs, have high potential to decrease WCET, but are one of the most difficult resources to time analyze [19]. In particular, *cache risk patterns* [20] (*crp*) are cache-induced sources of execution time variability difficult to capture in the WCET estimates. While nowadays automotive software does not use caches extensively, caches are key enablers of future automotive application's required performance (e.g., for autonomous driving), and need to be made timing-analysis friendly.

The most extended timing analysis practice relies on collecting software's execution time on the target hardware under different stressing conditions taking the high water mark (*hwm*) across all runs. In order to account for the uncertainty, an engineering factor is added to the *hwm*. Triggering the worst conditions in the presence of complex hardware involves deriving tests in which architectural low-level factors such as memory placement and the resulting cache layouts are exercised in stressing conditions. Controlling these micro-architectural effects is usually beyond the reach of end users, which reduces the confidence on the *engineering factor* and hence, on the WCET estimates.

Measurement-based *probabilistic* timing analysis (MBPTA) [9] has emerged recently to deal with complex hardware features and provide WCET estimates with increased confidence. To that end MBPTA combines i) time-randomized hardware features (such as caches) whose bad timing behaviours naturally emerge as more measurements are taken; and ii) probabilistic/statistical analysis that allows modelling the probability of several events to align resulting in high execution time. For the cache, MBPTA usually builds on top of random placement [16] (combined with random-replacement) that maps addresses randomly to cache sets across program runs. This allows deriving the probability of capturing at analysis time *crp* in which a high number of objects are mapped to the same set. Further, random placement provides independence between the address of objects in memory and their cache set placement, so the end user only needs to control the number of runs (tests) to perform – dictated by MBPTA – but not how program objects are allocated in memory – and the resulting cache layouts – in each run [7]. This is in contrast to conventional deterministic cache placement (either modulo or based on any fixed hash function [12, 25]) in which *crp* are triggered by the way in which program's data and code are loaded in memory – to a large extent out of the control of the end user – which may lead to arbitrarily high execution times not accounted for in the WCET estimates. In this context, this paper makes the following contributions:

① We make an in-depth analysis of the existing Random-Placement policy that is based on a hash function (hRP). Our results show that hRP, even when a program uses few contiguous cache lines, those lines can be (randomly) mapped to the same cache set with a non-negligible probability, which increases WCET estimates. While this problem does not happen in conventional modulo-based caches, since contiguous addresses are mapped to different sets, modulo-placement caches are not amenable to MBPTA.

② We propose Random Modulo (RM), an alternative ran-

DOI:

dom cache design suitable for MBPTA. The proposed design employs a new random placement policy that retains most of the advantages of modulo placement by avoiding conflicts across contiguous addresses while still meeting the requirements of MBPTA. With the proposed cache design the placement of objects is randomized by performing a random permutation of the original index bits in such a way that consecutive addresses are placed in non-adjacent positions in the cache but cannot create *crp* as long as cache capacity is not exceeded. By retaining spatial locality *RM* makes WCET estimates to be only slightly worse than average execution time values.

③ We perform, for the first time, an ASIC evaluation and FPGA implementation on a LEON3 processor of both *hRP* and *RM* random placement policies.

Our results show that *RM* achieves MBPTA compliance for which we confirm that probabilistic properties assessed with the corresponding independence and identical distribution tests [9] are passed. In terms of WCET estimates obtained with MBPTA for EEMBC Automotive benchmarks [21], results with *RM* improve those with *hRP*. In particular, *RM* obtains WCET estimates 43% tighter on average (and up to 62%) than *hRP*. ASIC-based results show that the *RM* module requires 10× lower area than that for *hRP*, and reduces 27% the delay of *hRP*. The same behavior has been observed in the FPGA prototype. Overall, *RM* contributes to the MBPTA ecosystem with a hardware low-complexity cache design that tightens WCET estimates, while keeping MBPTA compliance properties. This ultimately helps enabling a wider use of cache in automotive microcontrollers.

2. BACKGROUND

In ISO26262 timing testing and verification is one of the goals of the software unit design and implementation stage. While ISO26262 does not enforce specific methods, in this paper we focus on measurement-based analysis – the most common practice in industry [27]. Despite potential benefits in WCET reduction of caches, they generate uncertainty since, in general, for end users is difficult control memory mapping and its associated impact on cache and execution time. This exacerbates in the presence of integrated environments (such as AUTOSAR), where software may easily come from different providers and is integrated incrementally. How the modules of one software supplier interact with other modules, user-level libraries and OS functions is hard to determine. This puts a serious burden on the end user to ensure that memory mappings leading to pathological cache layouts (i.e. cache layouts resulting in high miss counts and execution time) are captured in the executions performed at analysis time. This difficulty is the reason why automotive microcontrollers, as a way to minimize timing verification costs, incorporate scratchpad memories like the AURIX [13], and/or allow configuring memory devices to be used as cache or RAM [14]. However, in the long-term enabling caches is of paramount importance to reduce WCET estimates and provide high guaranteed performance due to scalability and portability concerns with scratchpads.

MBPTA for Automotive Systems. MBPTA process starts by collecting several execution time measurements – typically in the order of few hundreds – of the program under analysis on top of a MBPTA-compliant hardware/software platform. Those measurements are used as input for Extreme Value Theory (EVT) [18], which is a powerful statistical method to approximate the tail of a distribution. In our case, the tail of the distribution corresponds to high execution times. Then, the probabilistic WCET (pWCET) is the execution time value of the obtained distribution that is exceeded with a given arbitrarily low probability. The exceedance probability is chosen to be low enough according to the application domain standards. For instance, if a program can be run up to 10 times per second (so up to 36,000 times per hour), we can use an exceedance threshold of 10^{-14} per run, which guarantees that failures per hour are below 10^{-8} as required by ASIL-D in ISO26262. Figure 1 shows an illustrative pWCET curve in which a cutoff probability of 10^{-7} (per hour) and the corresponding pWCET estimate are selected. Note that the pWCET curve is shown as a complementary cumulative distribution function (CCDF).

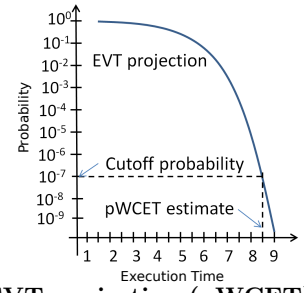


Figure 1: EVT projection (pWCET) in log. scale

MBPTA has been evaluated on multicores comprising last-level caches and shared buses, shared memory controllers [15, 24]. Further MBPTA has been positively assessed in the context of avionics case study [26].

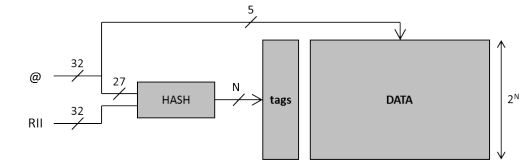
3. RANDOMIZED MODULO

All random placement policies build on top of a pseudo-random number generator (prng) to generate a new (pseudo-random) seed on demand. In [5] authors present a *prng* that i) provides enough randomness to pass MBPTA probabilistic tests; ii) has low implementation costs; and iii) is compliant with SIL3 in IEC-61508 (which ISO26262 bases on). In this paper we use and implement this *prng*.

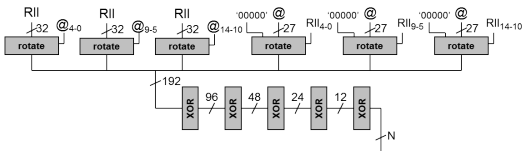
Every time the program is executed a new *seed* is generated leading to a new random mapping function corresponding to a cache layout. Different cache layouts cause different cache conflicts among memory addresses, resulting in different execution times. The number of possible cache layouts is given by S^u, where S is the number of sets and u the number of distinct memory addresses. On every *seed* change, the cache sets to which the addresses are mapped change as well, so cache contents must be flushed for consistency purposes. The exact point at which cache contents are to be flushed has to be determined considering the overhead due to cache flushing and the desired unit of analysis (timing entity). A simple approach that minimizes the overhead due to cache flushing is run to completion, which has been regarded as convenient in the avionics domain [26] and can be applied seamlessly at the granularity of runnables, tasks or software components in automotive.

3.1 Hash-Based Random Placement (hRP)

hRP is the only existing hardware random placement implementation meeting MBPTA requirements [16]. *hRP*, which uses a parametric hash function to determine the index to ac-



(a) Cache with *hRP*



(b) XOR-based Hash function

Figure 2: Hash-function RP (hRP) implementation.

cess cache sets, produces a unique index for a given memory address and random seed. *hRP* maps addresses to sets with homogeneous probabilities so that an address is mapped to a particular set with probability $1/S$, where S stands for the number of cache sets. In the context of MBPTA, an homogenous address-to-set distribution is desired although not strictly necessary. The only requirement is that the mapping is carried out probabilistically, so an address has a given probability to be mapped to any set – although that probability can change across sets – and the probability distribution at analysis time matches that during operation. In general, uniformly mapping address to cache sets typically reduces cache conflicts and hence pWCET estimates.

Implementation. Figure 2 shows the hash function integration in the cache (a) and a possible implementation of the parametric hash function (b). The hash function is placed just before accessing the cache memory contents (both data and tags) and uses all address bits except the offset ones to generate the index that is used to access the cache. For 32-bit addresses and 32-byte cache lines, the 5 lowest bits are discarded. Assuming a S-entry cache, the hash receives the 27 upper address bits and a random *seed* (referred to as *RII*) to generate an index of $N = log_2 S$ bits. The hash function is constructed using rotate blocks and 2-input XOR-gates to combine the output of rotate blocks to generate a cache index. In order to minimize cache conflicts, the hash uniformly maps addresses to cache sets.

In-depth Analysis of *hRP*. Programs exploit spatial locality by accessing code and data stored nearby recently accessed code and data such as, for instance, the instructions in a loop (typical in control-like applications) and the access to the stack (e.g. local variables) in a function.

Interestingly, with modulo placement, when those data/code are smaller than the cache way size, they do not collide in the same set, since addresses are mapped to consecutive cache sets. However, *hRP* gives each address the same probability to go to any set, so even for small address footprints there is a non-null probability that a subset of them (even all them) go to the same set. The net result is an increase in the number of potential cache conflicts that can occur with non-negligible probability and so an increase in WCET. With modulo placement, whenever addresses accessed are consecutive and they span across several cache ways but still fit in cache, they can be still accommodated in cache reducing the number of conflicts. Meanwhile, with *hRP* cache

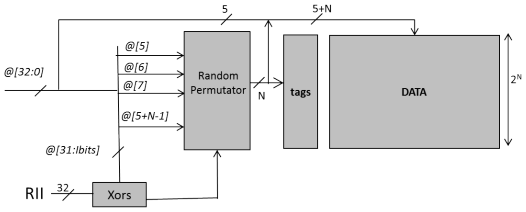


Figure 3: Random Modulo (RM) implementation.

conflicts increase with the size of the accessed data, even if the data fit in cache. Only when data do not fit in cache modulo placement can lead to systematic *crp* for consecutive memory addresses, whereas *hRP* may perform better with meaningful probability [23].

At implementation level, modulo placement does not need to store index bits to select the cache set since they are known implicitly. However, this relation is not preserved for *hRP*, which hence needs to store index bits in the tag array. Another drawback due to introducing the parametric hash function is the potential impact it has on the processor's critical path as all accesses to the cache go through the parametric hash function before accessing the cache, and such function consists of a level of rotate blocks and a cascade of XOR-gates of non-negligible depth (see Figure 2).

3.2 Random Modulo (RM) Placement

RM placement aims at obtaining the randomization properties of *hRP* while exploiting spatial locality as the modulo placement function does. *RM* exploits spatial locality by removing conflicts among addresses belonging to the same *cache segment*: Let CW_b be the size in bytes of a cache way and let assume cache comprising $S = 8$ sets, $W = 2$ ways and with a cache line of 32 bytes. This results in $CW_b = 8 \times 32 = 256$ bytes. All addresses that have the same cache-way alignment are said to belong to the same cache segment. That is, given two addresses A and B, if $\lfloor A/CW_b \rfloor = \lfloor B/CW_b \rfloor$ they are in the same cache segment.

Let us now assume that addresses A and B belong to the same cache segment and that with modulo are mapped to different sets k_A and k_B, respectively. *RM* creates a randomization of the index bits such that (in every run) with a seed $seed_i$, A is mapped to any (random) set $l_A = set_{rm}^{seed_i}(A)$ and B to $l_B = set_{rm}^{seed_i}(B)$ and l_A and l_B are necessarily different:

$$set_{mod}(A) \neq set_{mod}(B) \ \wedge$$
$$\lfloor A/CW_b \rfloor = \lfloor B/CW_b \rfloor \ \rightarrow \ set_{rm}^{seed_i}(A) \neq set_{rm}^{seed_i}(B) \quad \forall \ seed_i$$

RM makes a random permutation of the address index bits that is driven by a combination of a random seed (changed across runs) and the upper bits of the address. This removes the dependence among memory mapping and cache layout, and further ensures that the index permutation covers probabilistically during the analysis phase cache conflicts that reflect the conflicts that can occur at operation. Moreover, unlike *hRP*, *RM* exploits spatial locality, improving average performance and pWCET estimates w.r.t. *hRP*.

Implementation. Figure 3 shows a schematic of the *RM* integration in a cache design. *RM* introduces a hardware component that performs the permutation of the N-bit index. An efficient permutation of an arbitrary length index can be performed using a permutation network. When using a 8-bit Benes network 20 bits are required to drive the

actual permutation of the index bits. These bits are obtained by combining the upper address bits and the bits of the random seed. In particular upper address bits include all address bits (32 in our case) excluding offset (5 in our case) and index ones (8 in our case). Address upper bits and the random seed can be combined in several ways as far as it is preserved that small changes in address upper bits lead to different index permutations. In our implementation we concatenate the 19 upper address bits with the uppermost bit of the random seed and XOR them with the following 20 bits of the random seed, thus introducing almost only the delay of a XOR gate in the critical path.

– *Delay reduction*: With *RM* the extra delay added to access the cache is drastically decreased w.r.t. *hRP*. With *RM* index bits go directly through the permutation network pass transistors and overall delay is mainly determined by how upper address bits and the seed bits are combined to drive the index bits permutation.

– *Area reduction*: *hRP* needs to store index bits in the tag array because any pair of addresses can be placed in the same set for some seeds. In the case of *RM* those addresses within the same cache segment cannot be placed in the same set by construction and, therefore, hit/miss outcome can be determined by comparing uppermost address bits only (excluding the 8 index bits in our case). As a result, if the cache is write-through, *RM* does not need to store index bits, as in regular modulo-indexed caches. However, for write-back caches, as dirty lines may need to be written back on an eviction index bits are also needed to build the complete address. Nevertheless, most processor designs targeting safety critical applications typically rely on the use of write-through first-level caches [4][6].

Applicability. With *RM* cache-segment alignment must be maintained between analysis and operation: all addresses fitting in a cache segment in the experiments carried out at analysis, must remain in a segment at operation. This can be easily achieved when the RTOS memory page (or simply page) size is a multiplier of the cache segment size. This is already assumed by MBPTA to ensure conflicts at operation have been considered during the analysis phase. Furthermore, the real-time operating system (RTOS) may typically move objects (e.g. program binaries, stack frames, etc.) of programs and libraries across pages, but without changing their alignment w.r.t. page boundaries, thus meeting the requirements of *hRP* and *RM*. In fact this assumption has already been proven compatible with complex avionics case studies [26].

RM can be safely used for first level caches whose way size (i.e. cache segment size) is typically equal or smaller than the page size. If the way size is larger than the page size, which may be the case for second level (L2) caches (e.g., 128KB, 256KB), then *RM* can be used if the RTOS preserves page alignment at that granularity. For instance, if the cache way size is k times larger than the page size, the RTOS should maintain the alignment of pages at $k \times page\_size$ bytes granularity to safely apply MBPTA. If this cannot be guaranteed by the RTOS, then *hRP* can be used in the L2 instead.

MBPTA compliance. *RM* achieves MBPTA-compliance by breaking the dependence among memory mapping and cache layouts. In every run, any of the potential cache layouts can be randomly selected with *RM*. This gives each cache layout a probability of occurrence as needed by MBPTA. Existing techniques can be used to ensure that enough runs are performed at analysis time so that probabilistic representativeness is achieved [1]. Moreover, as shown in the results section, *RM* also passes independence and identical distribution (i.i.d.) tests as required to apply EVT.

Table 1: ASIC & FPGA implementation results.

	Area		Delay/Frequency	
	RM	hRP	RM	hRP
ASIC 45nm TSMC	$336.6um^2$	$3514.7um^2$	0.46ns	0.59ns
FPGA Stratix IV	72% occupation	80% occupation	100Mhz	80MHz

4. EVALUATION

In this section we show results obtained with an ASIC evaluation and a FPGA implementation of both *hRP* and *RM*. Many different processor architectures with different ISA have been found suitable for high-integrity automotive applications [13, 14]. For the evaluation of our proposal we choose a freely available RTL implementation of the LEON3 processor architecture that allows us to achieve a high technology readiness level. LEON3 has both instruction and data private 16KB 4-way L1 caches. We feature a 4-way 128KB shared L2 cache partitioned across the 4 cores.

We have used EEMBC Automotive Benchmark Suite [21] to analyze the performance that can be achieved with the proposed cache design. This is a well-known suite that mimics some real-world automotive critical functionalities. Further, in order to better analyze the benefits of *RM* over *hRP* we have also used a synthetic kernel application that accesses a vector with a data footprint that we have varied to (i) make it fit in L1 (8KB), (ii) not to fit in L1 but to fit in L2 (20KB), and (iii) not to fit neither in L1 nor in L2 (160KB). These benchmarks traverse the whole vector in a loop 50 times.

4.1 Implementation Results

We present implementation results in two flavors: ASIC-based area and delay synthesis results for *hRP* and *RM* modules in isolation; and FPGA-based results of area overhead and maximum operating frequency in a Stratix-IV FPGA prototype. Implementation results are shown in Table 1.

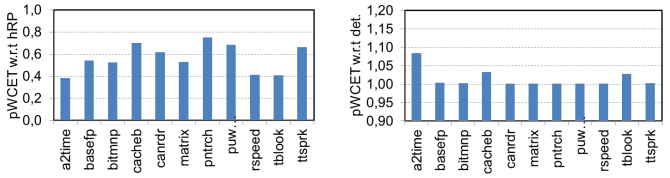
ASIC. For the ASIC results we have synthetized *RM* and *hRP* modules required to perform the random placement in a 128-set cache analogous to the instruction cache of the targeted processor. The synthesis is carried out using a 45nnm technology library from TSMC with Synopsys DC. ASIC-based results show that the *RM* module requires 10× lower area than the *hRP* one, and reduces around 27% the delay of the parametric hash function.

FPGA. In the FPGA prototype we compare the result of integrating either *RM* or *hRP* in all cache memories in the processor (IL1 and DL1 in each of the 4 cores, and the shared L2). The same trend as for the ASIC evaluation is observed in the case of the FPGA prototype. The baseline design is synthetized to operate at 100MHz, the maximum achievable frequency of the prototype. The delay introduced by *hRP* penalizes the targeted operating frequency, which needs to be decreased down to 80MHz. In terms of area occupancy, integrating *hRP* in all caches increases logic occupancy from 70% to 80%. With *RM* no degradation of the board operating frequency is required. Further *RM* increases the logic occupancy only by 2 percentage points, from 70% to 72%.

As explained in the implementation part in Section 3.2, area and delay reduction come from the fact that *RM* uses passgates mostly, whereas *hRP* requires more complex logic.

Table 2: WW and KS results for EEMBC benchmarks, which are identified by their initials.

	A2	BA	BI	CB	CN	MA	PN	PU	RS	TB	TT
WW	0.04	0.63	0.12	1.25	0.03	0.04	0.4	0.04	1.4	0.2	0.04
KS	0.9	0.06	0.05	0.08	0.46	0.13	0.06	0.06	0.05	0.09	0.76



(a) RM vs hRP (b) RM vs DET

Figure 4: RM pWCET results normalized to hRP results(a); and *hwm* in a deterministic setup (b).

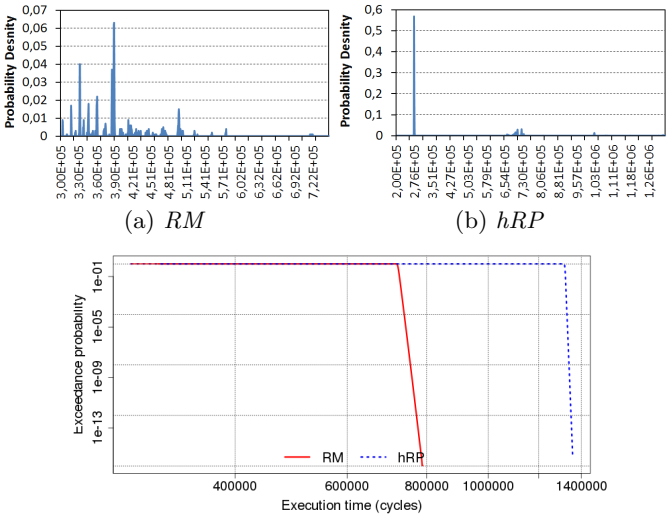
4.2 MBPTA compliance

We assess MBPTA compliance by checking whether the execution time observations obtained when RM is used, are i.i.d. Each EEMBC benchmark was run 1,000 times with different seeds collecting an execution time measurement in each case. We apply the Wald-Wolfowitz (WW) independence test [9] and the two-sample Kolmogorov-Smirnov (KS) identical distribution test [9]. For a 5% significance level (a typical value for this type of tests) for the WW test results below 1.96 prove independence. For the KS values above 0.05 indicate identical distribution. Table 2 shows that both tests are passed corroborating the MBPTA compliance of *RM*. We also applied and passed the ET [11] test for Gumbel convergence testing.

4.3 pWCET estimates: RM vs hRP

Following the MBPTA application methodology [9], we derived pWCET curves when running EEMBCs on the FPGA board. We consider two hardware setups: (1) DL1 and IL1 implementing *hRP* and (2) DL1 and IL1 implementing *RM*. For the L2 we use *hRP* in all cases. Figure 4(a) shows the pWCET estimates obtained for *RM* w.r.t. those for *hRP*. In particular we compare *hRP* and *RM* for a cutoff probability of 10^{-15} as it is valid for the highest criticality levels in both avionics and automotive. Similar results are obtained for 10^{-12} that can be used for functions not in the highest criticality levels. We observe that *RM* consistently provides tighter pWCET estimates than *hRP* for all EEMBCs ranging from 62% tighter pWCET estimates for `a2time` to 25% tighter for `pntrch`, with an average reduction of 43%.

Understanding the benefits of *RM* over *hRP*. As presented in Section 3, unlike *hRP, RM* exploits spatial locality resulting in a lower number of misses than *hRP* since consecutive memory addresses are ensured not to be mapped to the same set. This prevents some *crp* that happen with *hRP* in which many program addresses can be mapped to the same set. This is better illustrated in Figure 5(a) and (b) that show the probability density functions of the execution times collected for the synthetic benchmark when it has a 20KB footprint. *RM* shows much lower variability than *hRP*. *RM* execution times do not exceed 720,000 cycles. Conversely, for *hRP*, although most measurements are in a similar range as for *RM*, there is a number of cases where many lines are mapped into few sets leading to abundant conflicts that increase execution time noticeably (beyond 1,200,000 cycles). A similar effect occurs when the footprint of the synthetic benchmark varies. For 8KB footprints the effect reduces since almost all data fits in cache,



(a) *RM* (b) *hRP*

(c) pWCET

Figure 5: Probability density functions and pWCET estimates for *hRP* and *RM*.

while for a 160KB footprint the effect is more prominent since the footprint exceeds the L2 cache partition (128KB).

The net result is that pWCET values are tighter for *RM* than for *hRP* as shown in Figure 5(c) for the 20KB synthetic kernel. pWCET estimates are obtained with the MBPTA method described in [9]. Due to the higher execution times occurring with *hRP*, the pWCET curve for *hRP* is far beyond that of *RM*, thus proving that *RM* allows obtaining much lower pWCET estimates regardless of the target exceedance probability threshold.

4.4 Deterministic baseline

WCET. MBPTA has already been compared against techniques such as Static Deterministic Timing Analysis (SDTA) and Measurement-Based Deterministic Timing Analysis (MBDTA) [2]. MBPTA pWCET estimates have been shown to be competitive with the WCET estimates derived with SDTA and MBDTA, while simplifying the analysis process [3]. A detailed comparison of MBPTA against other timing analysis techniques is beyond the scope of this paper. For completion purposes, in this paper we compare MBPTA with a common industrial practice in safety-critical systems. This approach has been used for single-core systems during decades. It consists in collecting the high-water mark *hwm* of the application running on the target platform. An upperbound to the program execution time is derived by multiplying the *hwm* by an engineering factor, usually set to 20% [26]. While this 20% factor has been shown to work for simple architectures, it has not a scientific basis and the use of caches challenges its confidence.

Figure 4(b) shows that pWCET estimates achieved with *RM* are never higher than 7% w.r.t. to the *hwm*. In particular only `a2time`, `cacheb` and `tblook` provide pWCET estimates between 1% and 7% higher than the *hwm*. All other benchmarks are below 1%.

Overall, *RM* provides low pWCET estimates and higher guarantees than current industrial practice for which the 20% engineering margin (on COTS hardware) does not rely on strong solid arguments like those provided on top of MBPTA-compliant architectures.

Average performance is important in real-time systems to optimize non-functional metrics such as power and energy. Our results (not depicted for space constraints) show that

RM is on average only 1.6% worse than modulo placement with a maximum degradation of 8%. Hence, RM provides competitive performance results of modulo while achieving the MBPTA compliance of *hRP* with significant reductions in WCET estimates.

5. RELATED WORK

Some attempts have been performed to remove pathological cache access patterns by using cache indexing functions other than the conventional modulo [12, 25]. However, the behavior of all those cache designs is fully deterministic, and therefore, whenever a given input set produces a pathological access pattern, it will happen systematically for such input set. Some cache designs implement random replacement on set-associative caches [22, 6, 4]. Unfortunately, they rely on deterministic placement functions, thus not being MBPTA-compliant. Some authors developed a user-level library to randomly allocate program's objects in memory across runs [17]. While software-only solutions can be adopted in a shorter term than hardware solutions, they deliver less tight pWCET estimates [17] and require recompiling software.

Some works [10] statistically model deterministic caches timing behavior: these works assume that the observed variability in execution time has a random nature given for instance by the 'probability' of each execution path. In our context, this refers to the probability that each execution path (and the cache behavior that it produces) has during operation. However, it is challenging for an end user – if at all possible – to assign probabilities to execution paths. This would require the user to determine how many times each execution path of the program would execute during the lifetime of all the vehicles implementing that functionality. Randomized caches instead create a true randomized behavior that remains unaltered at analysis and operation making that analysis-time observations can be used to probabilistically model operation time behavior [7]. In that line, to the best of our knowledge, only *hRP* enables MBPTA-compliance for set-associative caches. In this paper we prove that our proposal, *RM*, outperforms *hRP* while still adhering to the requirements of MBPTA.

6. CONCLUSIONS

Cache impact on program's timing behavior is twofold. On the one hand, cache can significantly reduce WCET estimates by reducing the number of costly off-chip accesses. On the other hand, however, cache challenges deriving tight and sound WCET estimates, increasing the burden on the user to perform stress tests that provide evidence that cache risk patterns are captured. In order to attack these problems, in the context of MBPTA, we introduce random modulo, a new MBPTA-compliant placement function that offers average performance close to that of modulo placement. Random modulo also drastically reduces WCET estimates, average performance and hardware overheads, including both area and delay, in ASIC and FPGA implementations with respect to hash random placement. It is also remarkable that WCET estimates obtained with random modulo are just 1.5% higher than the actual execution time on COTS hardware, hence below the typical 20% engineering factor used by industry.

Acknowledgments

The research leading to these results has received funding from the European Community under the FP7 PROXIMA Project grant agreement no 611085, from the Ministry of Science and Technology of Spain under contract TIN2015-65316-P and the HiPEAC Network of Excellence. Carles Hernández is jointly funded by the Spanish Ministry of Economy and Competitiveness (MINECO) and FEDER funds through grant TIN2014-60404-JIN. Jaume Abella has been partially supported by the MINECO under Ramon y Cajal postdoctoral fellowship number RYC-2013-14717.

7. REFERENCES

[1] J. Abella et al. Heart of gold: Making the improbable happen to extend coverage in probabilistic timing analysis. In *ECRTS*, 2014.
[2] J. Abella et al. On the comparison of deterministic and probabilistic WCET estimation techniques. In *ECRTS*, 2014.
[3] J. Abella et al. WCET analysis methods: Pitfalls and challenges on their trustworthiness. In *SIES*, 2015.
[4] Aeroflex Gaisler. *Quad Core LEON4 SPARC V8 Processor - LEON4-NGMP-DRAFT - Users Manual*, 2011.
[5] I. Agirre et al. IEC-61508 SIL3 compliant pseudo-random number generators for probabilistic timing analysis. In *DSD*, 2015.
[6] ARM. ARM Cortex-R Series Processors Specification. http://infocenter.arm.com/help/topic/com.arm.doc.set.cortexr/index.html.
[7] F.J. Cazorla et al. Upper-bounding program execution time with extreme value theory. In *WCET Workshop*, 2013.
[8] R.N. Charette. This car runs on code. In *IEEE Spectrum online*, 2009.
[9] L. Cucu-Grosjean et al. Measurement-based probabilistic timing analysis for multi-path programs. In *ECRTS*, 2012.
[10] Yun Liang et al. Cache modeling in probabilistic execution time analysis. In *DAC*, 2008.
[11] M. Garrido and J. Diebolt. The ET test, a goodness-of-fit test for the distribution tail. In *MMR*, 2000.
[12] A. González et al. Eliminating cache conflict misses through XOR-based placement functions. In *ICS*, 1997.
[13] Infineon. AURIX - TriCore datasheet. highly integrated and performance optimized 32-bit microcontrollers for automotive and industrial applications, 2012.
[14] Texas Instruments. *SoC Processor for Advanced Driver Assist Systems*, http://www.ti.com/product/tda2 edition.
[15] J. Jalle et al. Bus designs for time-probabilistic multicore processors. In *DATE*, 2014.
[16] L. Kosmidis et al. A cache design for probabilistically analysable real-time systems. In *DATE*, 2013.
[17] L. Kosmidis et al. Probabilistic timing analysis on conventional cache designs. In *DATE*, 2013.
[18] S. Kotz and S. Nadarajah. *Extreme value distributions: theory and applications*. World Scientific, 2000.
[19] B. Lesage et al. WCET analysis of multi-level set-associative data caches. *WCET Workshop*, 2009.
[20] E. Mezzetti et al. Attacking the sources of unpredictability in the instruction cache behavior. In *RTNS*, 2008.
[21] J. Poovey. *Characterization of the EEMBC Benchmark Suite*. North Carolina State University, 2007.
[22] E. Quinones et al. Using randomized caches in probabilistic real-time systems. In *ECRTS*, 2009.
[23] M. Slijepcevic et al. DTM: Degraded test mode for fault-aware probabilistic timing analysis. In *ECRTS*, 2013.
[24] M. Slijepcevic et al. Time-analysable non-partitioned shared caches for real-time multicore systems. In *DAC*, 2014.
[25] Z. Wang and R.B. Lee. A novel cache architecture with enhanced performance and security. In *MICRO*, 2008.
[26] F. Wartel et al. Timing analysis of an avionics case study on complex hardware/software platforms. In *DATE*, 2015.
[27] R. Wilhelm et al. The worst-case execution time problem: overview of methods and survey of tools. *In TECS*, 7, 2008.