Applying Measurement-Based Probabilistic Timing Analysis to Buffer Resources

Leonidas Kosmidis[1,2], Tullio Vardanega[3], Jaume Abella[2], Eduardo Quiñones[2], and Francisco J. Cazorla[2,4]

1 **Universitat Politècnica de Catalunya**
2 **Barcelona Supercomputing Center**
3 **University of Padova**
4 **Spanish National Research Council (IIIA-CSIC)**

Abstract

The use of complex hardware makes it difficult for current timing analysis techniques to compute trustworthy and tight worst-case execution time (WCET) bounds. Those techniques require detailed knowledge of the internal operation and state of the platform, at both the software and hardware level. Obtaining that information for modern hardware platforms is increasingly difficult.

Measurement-Based Probabilistic Timing Analysis (MBPTA) reduces the cost of acquiring the knowledge needed for computing trustworthy and tight WCET bounds. MBPTA based on Extreme Value Theory requires the execution time of processor instructions to be independent and identically distributed (i.i.d.), which can be achieved with some hardware support. Previous proposals show how those properties can be achieved for caches. This paper considers, for the first time, the implications on MBPTA of using buffer resources. Buffers in general, and *first-come first-served (FCFS) buffers* in particular, are of paramount importance as the complexity of hardware increases, since they allow managing contention in those resources where multiple requests may be pending. We show how buffers can be used in the context of MBPTA and provide illustrative examples.

1998 ACM Subject Classification D.2.4 Software Engineering: Software/Program Verification

Keywords and phrases WCET, Buffer, Probabilistic Timing Analysis

Digital Object Identifier 10.4230/OASIcs.WCET.2013.97

1 Introduction

There is an increasing need for high guaranteed performance in Critical Real-Time Embedded Systems (CRTES) industry such as automotive, space, and aerospace. To respond to this demand, more complex hardware is used, which allows increasing performance per chip unit, which in turn enables running more functionalities per chip, thus reducing size, weight and power consumption costs at system level.

Probabilistic Timing Analysis (PTA) [4][3] has recently emerged as an alternative to conventional static (STA) and measurement-based timing analysis (MBTA) techniques [11]. Although PTA is not as mature as STA and MBTA yet, it promises to reduce dependence on execution history. This is done by randomising the timing behaviour of some processor resources, which reduces the amount of information needed to obtain tight WCET bounds in comparison to other timing analysis approaches.

PTA provides WCET estimates with an associated probability of exceedance (pWCET). In analogy to the practice that expresses reliability for embedded safety-critical systems in terms

13th International Workshop on Worst-Case Execution Time Analysis (WCET 2013).
Editor: Claire Maiza; pp. 97–108

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of allowable probabilities of hardware failures, PTA extends this notion to timing correctness by determining the probability with which a given WCET bound can be exceeded during system operation. PTA aims to obtain pWCET estimates for arbitrarily low probabilities, so that even if the chosen pWCET estimate can be exceeded, it would be with low enough probability (e.g., in the region of 10^{-12} per hour of operation, largely below the required probability of hardware failures). PTA can be applied either in a static (SPTA) [3] or measurement-based (MBPTA) [4] manner. This paper focuses on the latter, which is more easily amenable to industrial practice.

Contribution. PTA can be applied to hardware/software platforms where the ETP per instruction can be derived. PTA-compliance has been achieved so far for processors equipped with cache memories [5, 6]. In this paper we extend this to buffer resources. Buffers allow managing contention in those resources where multiple requests may be pending, decoupling the speed at which requests are sent and processed. Our contribution is threefold: (1) We prove that buffers can be used while preserving compliance with MBPTA requirements. Unlike other resources like caches that need to be time-randomised in order to work properly with MBPTA, buffers require no changes to be used with MBPTA. (2) We provide a new classification of hardware resources and describe how they can be adopted with MBPTA. (3) We show that, although buffers and any other complex resource in general can create dependences across instructions, they can be analysed by MBPTA as long as those dependences, regardless of their nature, whether deterministic or probabilistic, stay the same at analysis and during operation. For buffers in particular, we show how the dependences they create across instructions are purely probabilistic in a MBPTA-compliant processor and do not change between analysis and deployment.

2 Background

Unlike previous analysis techniques that provide a single WCET value per program, PTA provides a distribution function that upper bounds the execution time of the program under analysis, guaranteeing its execution time only exceeds the corresponding execution time bound with a probability lower than a given target threshold (e.g., 10^{-16} per activation). In this way the pWCET is defined as the execution time bound with its associated exceedance probability.

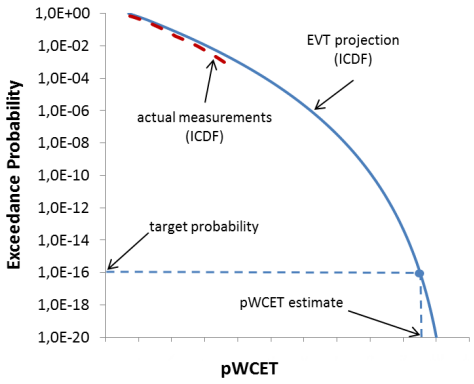


Figure 1 Example of the pWCET curve.

The timing behaviour of a program (and equivalently that of individual processor instructions) is represented with an Execution Time Profile (ETP). An ETP is the probability distribution function describing the different execution times that the program can take (the latencies, for processor instructions) and their associated probabilities. That is, the timing behaviour of a unit of execution (program, instruction) can be defined by the pair of vectors $(\vec{l}, \vec{p}) = \{l_1, l_2, ..., l_k\}\{p_1, p_2, ..., p_k\}$, where p_i is the probability the program/instruction having latency l_i with $\sum_{i=1}^{k} p_i = 1$.

The ETP for a program (resp. instruction) may vary with the program input sets that lead to different execution paths. Furthermore, the ETP for an instruction may vary

across multiple uses as execution events (e.g. previous accesses to memory) affect the state-dependent timing behaviour of that instruction. In Annex I we analyse those aspects showing that (1) the effect of past random events affect the ETP of an instruction in a probabilistic manner, whereby PTA continue to be applicable; and (2) each PTA technique has its own mechanisms to address the multiple execution path problem.

MBPTA requires the hardware to guarantee that each operation (at the granularity of processor instructions or below) has its own ETP. However, unlike SPTA, which needs all ETPs to be known, MBPTA only requires those ETPs to exist. In other words, if execution times were collected by rolling a die, SPTA would need to know the number of faces of that die, the value on those faces, and their individual probabilities of occurrence. Conversely, MBPTA would derive pWCET estimates by simply rolling the die, that is to say, by executing the program a given number of times, observing the resulting execution times and treating them with *Extreme Value Theory* (EVT) [4, 8] to a trustworthy and tight upper bound to the tail of the observed execution time distribution. By doing so, MBPTA provides *pWCET* estimates for arbitrarily low *target probabilities*. Figure 1 shows a hypothetical result of applying EVT to a collection of 1,000 observed execution times. The dotted line represents the inverse cumulative distribution function (ICDF) derived from the observed execution times. The continuous line represents the projection obtained with EVT.

2.1 Probabilistic Behaviour of Simple Processor Resources

Processor resources can be regarded as abstract components that process requests. Each such request has a distinct service time or latency, which can either be fixed or variable.

Jitter-centric resource classification. We term *jitter* the difference between the best and worst possible latency of any resource. Resources can be then classified depending upon whether they exhibit jitter or not. *Jitterless resources* have a fixed latency, independent of the input request or of the past history of service of the resource. Many hardware resources in current processor architectures can be classified as jitterless. Other resources, for instance cache memories, have a variable latency and hence are *jittery resources*; their latency depends on their history of service, i.e., the execution history of the program, the input request, or a combination of both. Jittery resources have an intrinsically variable impact on the WCET estimate for a given program. The significance of this impact depends on the magnitude of the jitter, the program under study, and the analysis method. A way to deal with jittery resources in the absence of timing anomalies is to assume that all requests to those resources *incur the worst-case latency* [9]. This is acceptable if the cumulative impact on the WCET from assuming the worst-case jitter for the resource is deemed low enough by the system designer. If taking the worst latency is not acceptable, then the timing behaviour of the resource must be randomised. This is the case of the cache, since taking its worst latency would greatly amplify the pWCET estimate. Several works propose time-randomising caches to reach both, probabilistically analysable behaviour and high guaranteed and average performance [5, 6].

ETP and jitter. Jitterless resources are easy to model for all types of static timing analysis. Building the ETP of a simple instruction that uses a single resource, requires knowing only whether the resource in question is jitterless (information implicit in the instruction) or whether the instruction is part of a sequence of instructions that must incur a delay when using a jitterless resource (information implicit in the architecture). With proper path and pipeline analysis, the types of the resources can be determined. Of course, measurements obtained from program runs that only use jitterless resources will perfectly capture their

Table 1 Code example with hit/miss probabilities for the instruction and data caches.

instruction id	instruction type	IL1		DL1	
		hit prob.	miss prob.	hit prob.	miss prob.
$i1$	LD	1.0	0.0	0.9	0.1
$i2$	ADD	0.7	0.3	-	-
$i3$	ADD	0.6	0.4	-	-
$i4$	ADD	1.0	0.0	-	-

constant impact on execution time. If the instruction accesses a jittery resource whose worst-case latency is acceptable for the designer, forcing that resource to always take the longest latency would be a simple yet effictive way to make the resource PTA-conformant: the ETP of that resource would have a single latency value (its worst case) with probability 1, i.e. 100% probability of maximum latency, leading to a upper-bounded deterministic jitter.

Instructions may access multiple resources during their execution, and those resources can be arranged in different manners, e.g. sequentially or in parallel. Under each arrangement, the ETP of those resources can be properly combined to derive the ETP of the instruction. To that end, several forms of convolution, $\otimes$ [3], can be used either adding latencies (sequential arrangements) or picking the maximum latency of the elements convolved (parallel arrangements).

3 Complex Processor Resources

However, the taxonomy presented in previous section does not cover *buffers*, which in fact are in widespread use in modern processor architectures. Buffers are used to temporarily hold some information decoupling the timing of the sender and the receiving elements. If a buffer is full it may create stalls that propagate backwards in the pipeline of the processor, thus potentially increasing the execution time and affecting WCET.

3.1 Timing Behaviour of a Buffer in a Time-Randomised Architecture

For the sake of illustration, let us assume an architecture with two stages (fetch and execute) that respectively access instruction and data caches (IL1 and DL1 for short). Caches deploy random placement and random replacement [5], which enable computing a probability of hit/miss for every access. In between both stages there is a 2-entry buffer (see Figure 2). In case of hit in both caches and if the buffer is available, an instruction takes 3 cycles: Fetch (F), buffer (b) and Execute (E).

Further assume that we execute the program with four instructions shown in Table 1, whose hit and miss probabilities for each cache are shown next to each instruction. For this example, $i1$ always hits in IL1 and has a 0.9 hit probability in DL1. The remaining instructions do not access DL1.

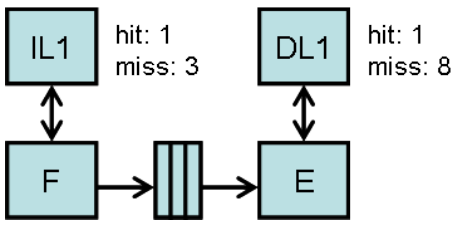


Figure 2 Processor setup considered in Section 3.1.

In the program fragment shown in Table 1, $i1$ may introduce some delay in the execution of the program when accessing DL1. In particular, if it misses in the data cache it will cause a longer delay than if it hits. Note that the IL1 hit probability of $i1$ is 100%, hence always hitting in IL1. $i2$ and $i3$ may introduce some delay when accessing IL1 only since they are not memory operations.

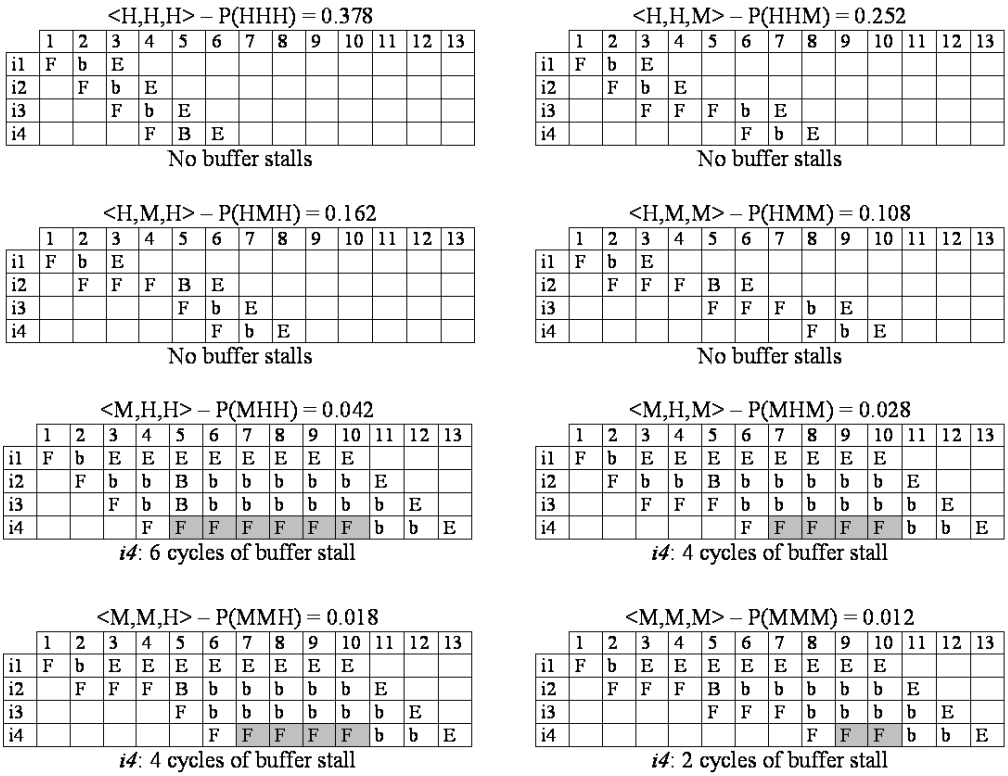


Figure 3 Potential chronograms based on the outcome of the different cache accesses. (<DL1-i1 IL1-i2 IL1-i3>) Grey rectangles show the cycles in which the processor is stalled due to the buffer.

In Figure 3 we depict the 8 different chronograms for each one of the combinations of hits and misses in IL1 and DL1 of all 4 instructions. The x-axis shows the cycles of execution while the y-axis shows each instruction. Each rectangle represents the stage in which each instruction is in each cycle: 'F' fetch, 'b' buffer and 'E' execute. We use the vector $<DL1\text{-}i1, IL1\text{-}i2, IL1\text{-}i3>$ to describe the outcome of each DL1 and IL1 access, being H a hit and M a miss. For instance <HHH> is the event 'i1 hits in DL1' and both 'i2 and i3 hit in IL1'. Similarly P(HHH) is the probability of that event to happen. Note that $i1$ and $i4$ have IL1 hit probability of 100% so for this reason IL1i1 and IL1i4 do not appear in the vector.

The key appreciation we do in the behaviour of the buffer is the following: given a set of fixed initial conditions (e.g. empty state of the pipeline) each different combination of probabilistic events (e.g. DL1 and IL1 accesses) leads to exactly one fully-deterministic behaviour of the buffer. If we compare different outcomes of probabilistic events, we observe that the buffer introduces a different number of stall cycles (0, 2, 4 or 6 cycles) for each combination of probabilistic events. The number of stalls and the particular cycles in which the stalls occur may repeat in different sequences of outcomes of the probabilistic events occurring (for instance cases <M,H,M> and <M,M,H>). However, for a particular sequence of random events the behaviour of the buffer is fully deterministic: all data dependences, which are given by the sequence of instructions that are executed and their order. Given that MBPTA works or a per-path basis, in each path the sequence of instructions executed is known and fixed across runs of the same path.

The initial conditions can be caused to a fixed state by flushing the state of the resource prior to its use. Alternatively, it might be possible to probabilistically determine the state

left by previously executing code. We refer the reader to [7]. for more details.

In order to better understand this phenomenon, Figure 4 depicts, for the same example shown before, the *probability tree* for the states of the processor in each cycle. In cycle 1 *i1* is fetched. In cycle 2 *i1* is stored in the buffer while *i2* is fetched. Accessing DL1 is a random event that has two outcomes hit/and miss, and hence spawns into two possible probabilistic states, which generates a new branch in the probability tree as shown in cycle 2.

In the left branch, during cycle 3, *i1* accesses DL1 while *i3* accesses IL1. Both are probabilistic events that generate 4 new branches in the probability tree. Similarly, in the right branch in cycle 3, *i1* accesses DL1 generating two branches in the probability tree.

As shown, the variability in the execution time increases the number of potential probabilistic states that we can reach. It is interesting noting that all the execution time variability can only be introduced by probabilistic events.

In this diagram, the stalls due to the buffer are shown with grey boxes. Unlike caches that introduce probabilistic variability, and hence generate new branches in the probability tree, buffer stalls cannot produce probabilistic variability, instead *buffer variability has no effect on the probability of each execution time to occur*. Therefore buffers *cannot create probabilistic jitter* but simply propagate jitter or, in other words, given a sequence of outcomes for all probabilistic events the delay of the buffer resources is fully deterministic.

Under MBPTA, the fact that buffer resources can affect the duration of the program under each combination of probabilistic events but cannot affect the probability of each combination, simplifies their analysis. As long as the execution time observations obtained sufficiently cover, in probabilistic terms, the outcome of random events, it is also enough to safely cover the effect of buffers.

3.2 Classification of Sources of Jitter

So far we regarded jitter as deterministic or probabilistic (the latter for time-randomised resources). Yet, as shown above, the jitter caused by buffers does not fit into either category; instead, it simply propagates the inbound jitter regardless of its nature.

With this insight, we classify the potential sources of jitter into 6 groups depending on the combination of two factors: (i) whether the jitter is produced solely by the event under consideration (no history dependence) or by the combination of previous events and the current one (history dependence); and (ii) whether the jitter is deterministic, probabilistic or simply propagated regardless of its source. We omit two groups for which we did not find any existing resource to fit in.

This new classification of hardware resources will help analysing whether a given resource or processor architecture is MBPTA-compliant. To that end, for each group we identify how resources in that group can be used in the context of MBPTA.

- *No history dependence + deterministic jitter.* This could be the case of a resource whose latency does not depend on the sequence of requests it has received, but on the data of each request. For instance, the floating-point unit in some processors is affected by the particular operands (data) being operated. For this type of resources we typically enforce the unit to experience always its maximum latency as explained before, which can be done deploying a simple hardware mechanism called the worst-case mode [9].
- *History dependence + probabilistic jitter.* This is the case of a time randomised cache [5]. The sequence of events between two consecutive accesses to the same data together with the initial cache state, determine the hit/miss probability of that access. Time randomised caches have been shown to be analysable with MBPTA [5].

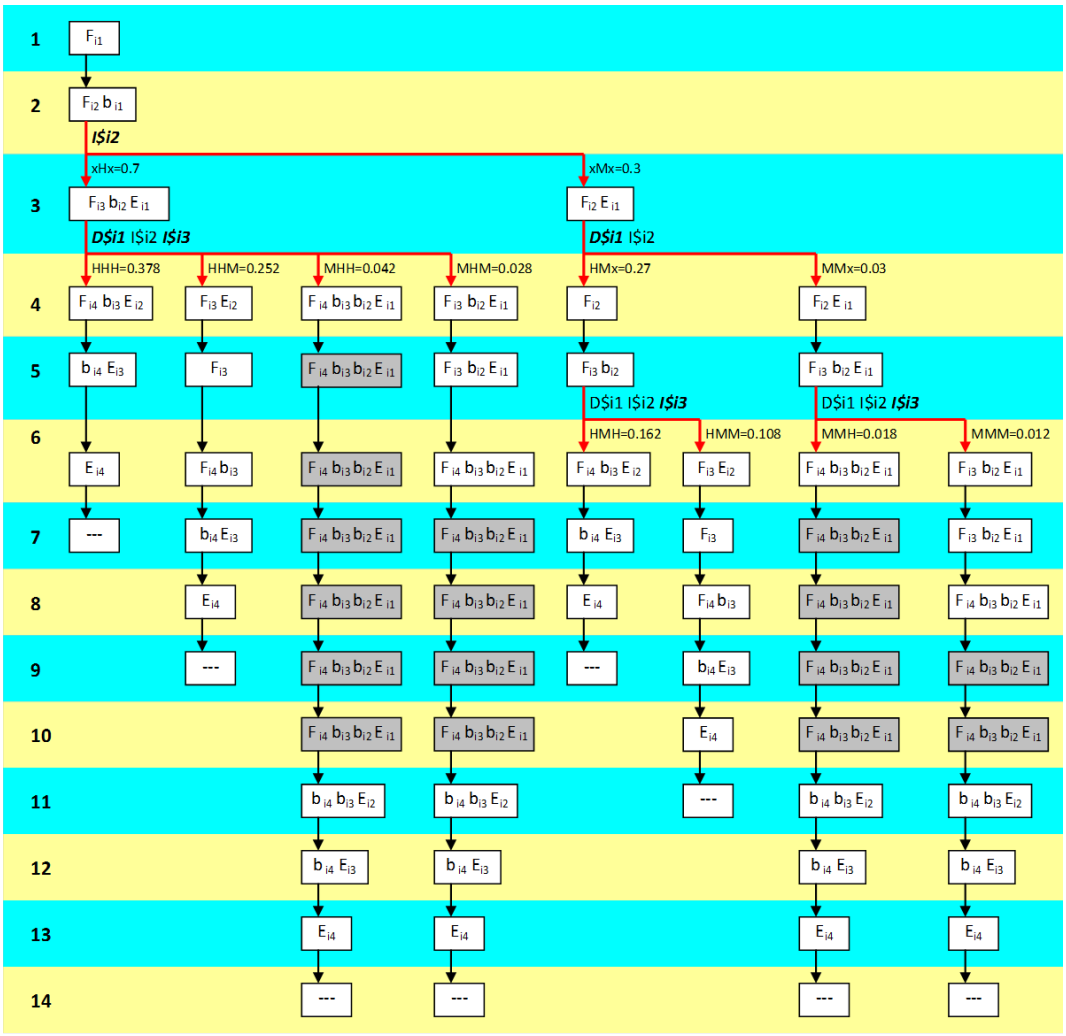


Figure 4 Processor Stage Graph.

- *History dependence + deterministic jitter.* This is the case of a deterministic cache implementing modulo placement and LRU replacement. Events may experience different latencies depending on previous history: for a given initial state and a sequence of events their latency is always the same. This type of resources is not analysable by MBPTA in general unless the factors that influence the jitter are fully under control, so that it can be known whether the observations taken to feed MBPTA cover the worst behaviour of those factors of influence. In general, the only easy way to enable the use of this type of resources in the context of MBPTA is using the worst-case mode.

- *History dependence + jitter propagation.* This is the case of a hardware buffer. A particular instruction may spend a different number of cycles in a buffer depending on previous events. However, as explained before, buffers do not create new jitter by themselves. Instead, they only propagate deterministically the effect of the jitter induced by other resources. If such jitter is probabilistic, then the stalls induced by buffers occur also with a given probability and so they are analysable with MBPTA.

3.3 Empirical Verification

Although we have described how buffers meet the MBPTA requirements if they are already fulfilled by the processor in use without buffers, in this section we verify empirically that this claim holds by testing that execution times in such a processor are independent and identically distributed, as required by MBPTA. To that end we apply the experimental methodology shown in [4].

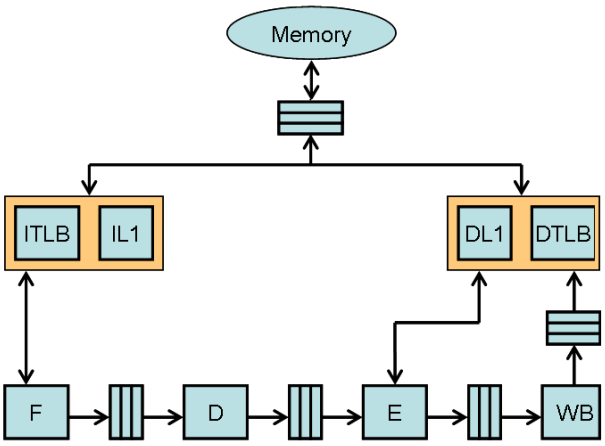


Figure 5 Processor setup considered in Section 3.3.

We consider a pipelined processor with in-order fetch, dispatch and retirement of instructions (see Figure 5). Fetch and execution stages are equipped with first level instruction and data cache memories respectively (IL1 and DL1 caches for short). Instruction and data translation look-aside buffers (ITLB and DTLB) are also in place. Buffers across pipeline stages are deployed to mitigate stalls. Similarly, a store buffer is provided to allow store instructions to retire quickly without stalling the pipeline[1]. Both IL1 and DL1 size are 4KB 8-way 16-byte line caches. Both caches implement random placement and replacement policies [5]. DTLB and ITLB are 16-way fully associative, and page size is 1KB. The latency of the fetch stage depends on whether the access hits or misses in the IL1 and ITLB: only if the access hits in both its latency is 1-cycle, and 100 cycles otherwise. After the decode stage, memory operations access the DL1 and DTLB and their behaviour is analogous to that of IL1 and ITLB. The remaining operations have a fixed execution latency (e.g. integer additions take 1 cycle).

For our experiments we use the EEMBC Autobench benchmark suite [10] that reflects the current real-world demand of automotive systems. The fact that, unlike EEMBC, real-world programs normally have multiple paths does not invalidate the conclusions of our analysis: this is so because MBPTA considers individual paths.

In order to test independence we use the Wald-Wolfowitz independence test [2]. We use a 5% significance level (a typical value for this type of tests), which means that absolute values obtained after running this test are below 1.96 if there is independence, otherwise are higher. For identical distribution, we use the two-sample Kolmogorov-Smirnov identical distribution test [1] as described in [4]. For a 5% significance level, the outcome provided by the test should be above the threshold (0.05) to indicate identical distribution, otherwise non-identical distribution.

Table 2 shows the results of both tests for all EEMBC benchmarks, when running each benchmark as many times as needed by MBPTA (up to 1,000 times per benchmark in our evaluation). As shown, both tests are passed in all cases.

[1] A store buffer is a particular incarnation of buffer resources.

Table 2 Independence and identical distribution tests results.

Benchmark	a2time	aifftr	aifirf	aiifft	cacheb	canrdr
Indep. test	0.90	0.10	0.27	0.11	0.51	0.21
Ident. distr. test	0.64	0.93	0.84	0.70	0.40	0.39

Benchmark	iirflt	puwmod	rspeed	tblook	ttsprk	
Indep. test	0.11	0.37	0.33	0.47	0.63	
Ident. distr. test	0.80	0.89	0.27	0.93	0.73	

4 Conclusions

In this paper we show that buffer resources do not create any jitter on their own but, instead, they simply propagate inbound jitter regardless of the nature of it. With this, we prove that buffers do not break PTA requirements, hence can be used in PTA-conforming processors with no change. We also provide a comprehensive classification of hardware resources and how they can be considered in the context of PTA.

Acknowledgements

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under the PROARTIS Project (www.proartis-project.eu), grant agreement no 249100. This work was partially supported by EU COST Action IC1202: Timing Analysis On Code-Level (TACLe). This work has also been partially supported by the Spanish Ministry of Science and Innovation under grant TIN2012-34557 and the HiPEAC Network of Excellence. Leonidas Kosmidis is funded by the Spanish Ministry of Education under the FPU grant AP2010-4208. Eduardo Quiñones is partially funded by the Spanish Ministry of Science and Innovation under the Juan de la Cierva grant JCI2009-05455.

A Annex I. Meeting MBPTA requirements

Next, we show how the existence of an ETP for each instruction in a program is a necessary and sufficient condition to make a program and a target platform analysable with MBPTA. To that end and without loss of generality we assume a processor architecture in which core operations (e.g., MUL and ADD) take a fixed latency and memory operations (e.g., LD and ST) access a fully-associative random-replacement cache [5].

Let us assume a fully-associative cache with W ways and random replacement[2]. An approximation to the probability of hit of a given access Aj in the sequence $< Ai\ B1\ B2, ...,\ Bk\ Aj >$, where Ai and Aj access the same cache line and all Bl access other cache lines, is given by [5]:

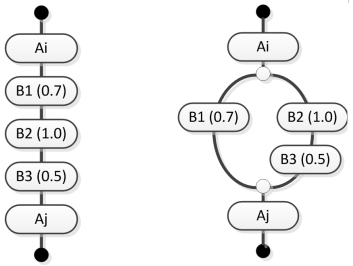
$$P_{hit_{A_j}}(S) = \left(\frac{W-1}{W}\right)^{\sum_{l=1}^{l=k} P_{miss_{Bl}}} \quad (1)$$

In the equation, $\frac{W-1}{W}$ is the probability of one access to evict Aj, while the exponent gives a measure of the number of evictions Aj can suffer depending on the probability of each $\{B_l\}_{l\in(1..k)}$ to miss in cache. We observe that Aj depends on execution history, i.e., $\{B_l\}_{l\in(1..k)}$. In particular, the probability of miss of $\{B_l\}_{l\in(1..k)}$ affects the probability of hit/miss of Aj. In a given run, the fact that a given Bl hits/misses in cache affects the probability of hit of Aj in that run. For instance the probability of hit of Aj in a run in which $B1$ misses is different from another run in which $B1$ hits. Hence, under each history of outcomes for $\{B_l\}_{l\in(1..k)}$, Aj may have a different hit probability.

We focus on two scenarios as depicted in Figure 6. In the first one, Figure 6(a), the whole sequence of accesses is in the same basic block, while in the second one, Figure 6(b), the sequence of accesses is spread across several basic blocks.

1) Single path. When all accesses in a sequence affecting a given access Aj are in the same basic block, they affect Aj in each run systematically, since all $\{B_l\}_{l\in(1..k)}$ are present in each run. Under each history of outcomes Aj may have a different probability of hit, and hence a different ETP.

Interestingly, hit/misses affecting Aj's probability of hit are random events by construction for a cache using random replacement and random placement. This introduces a probabilistic variation in the probabilities of each execution time of Aj. Hence, if enough runs are made the observed frequencies of hit/miss for each Bi and Aj will converge to their actual hit/miss probability. As a consequence, Aj can be regarded as having an ETP, where the probability of hit of Aj is that resulting from executing the program an infinite number of times. In the example in Figure 6(a), for a cache with W=8 ways the ETP of A_j is as follows: $\{l_h, l_m\}\{0.745, 0.255\}$.



(a) single basic block (bb) (b) branch structure (several bb)

Figure 6 Cache access sequences and distribution over different basic blocks.

Note that, if the variability that $\{B_l\}_{l\in(1..k)}$ causes on A_j is not probabilistic, which happens for instance if cache is not randomised (e.g., if modulo placement is used), the hit

[2] A similar analysis can be done for set-associative caches [5].

Table 3 ETPs of Aj under each history of outcomes.

B1-3 outcome hist.	No. of Evicts.	Prob. of that outcome	ETP of (Aj) under that outcome history
000	3	0.35	$\{l_h, l_m\}\{0.670, 0.330\}$
001	2	0.35	$\{l_h, l_m\}\{0.766, 0.234\}$
010	2	0	-
011	1	0	-
100	2	0.15	$\{l_h, l_m\}\{0.766, 0.234\}$
101	1	0.15	$\{l_h, l_m\}\{0.875, 0.125\}$
110	1	0	-
111	0	0	-

event for $\{B_l\}_{l\in(1..k)}$ is not random, so we could not derive an ETP for Aj disallowing the use of MBPTA.

Results. In the example in Figure 6(a) there is a dependence between Aj and the history of outcomes of $B1, B2$ and $B3$. In particular, for a given run the number of misses incurred by $B1$-$B3$ determines the number of random evictions carried out between Ai and Aj. The second column in Table 3 shows the number of evictions carried out under each outcome history for $B1$-$B3$. The third column shows the probability of that outcome based on the probability of miss of $B1$-$B3$. Finally, the fourth column shows the ETP of Aj assuming a fully-associative cache of 8 ways. With enough runs, the final miss probability for Aj can be computed as the addition of the probability of each possible history of outcome of $B1$-$B3$ times the probability of Aj to miss under that outcome:

$$P^m_{Aj} = \sum_{k=1}^{No.Outcomes} Prob_{Outcome_k} \times P^m_{Aj^{Outcome_k}} \quad (2)$$

that in our example results in: $P^m_{Aj} = (0.330 \times 0.35) + (0.234 \times 0.35) + (0.234 \times 0.15) + (0.125 \times 0.15)$=0.251, that accurately matches the value computed with Equation 6[3], where $P^h_{Aj} = (7/8)^{(0,7+1+0,5)}$. Hence the ETP for Aj is: $\{l_h, l_m\}\{0.749, 0.251\}$.

Therefore, although the probability of each latency of an instruction depends on its execution history – the set of outcomes of previous accesses in our case – the fact that factors of influence on its execution history are random, and hence they occur with a given probability, makes it possible to derive an ETP for the instruction. If enough samples are taken from the timing behaviour of Aj during analysis time, the observed behaviour is representative of its behaviour during deploy time. This is so because the factors of influence on Aj execution time have a random nature, so for a higher number of runs the observed frequencies of each event converge to the actual probability of the event.

2) Multiple paths. In the situation depicted in Figure 6(b) we observe that the hit probability of Aj depends on the particular path followed. Hence, the ETP of Aj is affected by: the path followed and the history of hit/misses. If Aj is reached through the left path, hence under

[3] Note that minor discrepancies are expected given that hit/miss events are not independent among them, so the hit probability computed in Equation 1 is an approximation. In fact, there are only two ways to derive the actual hit/miss probabilities: (i) Performing an infinite number of runs and measure actual probabilities, or (ii) Computing the probability of each particular cache state left by the sequence of hits and misses for previous accesses, and accumulating the probabilities for those cache states where the current cache access would result in a hit/miss.

the sequence $< Ai\ B1\ Aj >$ it has higher probability of hit than if it is reached through the right path under the sequence $< Ai\ B2\ B3\ Aj >$: $ETP_{left} = \{l_h, l_m\}\{0.911, 0.089\}$ and $ETP_{right} = \{l_h, l_m\}\{0.818, 0.182\}$. Differently to the single-path case, now there is one ETP per path leading to Aj. MBPTA provides pWCET estimates for the set of paths exercised with the input data used during the testing phase. It is also the case that MBPTA is insensitive to the frequency each path is exercised as long as each path is exercised a minimum number of times [4]. Overall, having for each instruction and path-leading-to-that-instruction one ETP preserves the i.i.d. property in the execution time of each path. MBPTA [4] samples the execution time observations obtained from each path to obtain an i.i.d. sample that covers the execution time observed for all paths.

Results. In [4] it is shown how MBPTA works for multipath analysis: If enough execution time observations are obtained under each path, the effect that Aj can suffer from any of the Bl in each path is captured in probabilistic terms. This is a sufficient condition for MBPTA to provide safe upper-bounds.

References

1 Sarah Boslaugh and Paul Andrew Watters. *Statistics in a nutshell.* O'Reilly Media, Inc., 2008.

2 J.V. Bradley. *Distribution-Free Statistical Tests.* Prentice-Hall, 1968.

3 F.J. Cazorla, E. Qui nones, T. Vardanega, L. Cucu, B. Triquet, G. Bernat, E. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo, and D. Maxim. Proartis: Probabilistically analysable real-time systems. *ACM TECS*, 2012.

4 L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzeti, E. Quinones, and F.J. Cazorla. Measurement-based probabilistic timing analysis for multi-path programs. In *ECRTS*, 2012.

5 L. Kosmidis, J. Abella, E. Quinones, and F.J. Cazorla. A cache design for probabilistically analysable real-time systems. In *DATE*, 2013.

6 L. Kosmidis, C. Curtsinger, E. Quinones, J. Abella, E. Berger, and F.J. Cazorla. Probabilistic timing analysis on conventional cache designs. In *DATE*, 2013.

7 L. Kosmidis, E. Quinones, J. Abella, T. Vardanega, and F.J. Cazorla. Achieving timing composability with measurement-based probabilistic timing analysis. In *In IEEE International Symposium on Object/component/service-oriented Real-time distributed computing (ISORC)*, 2013.

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

9 M. Paolieri, E. Quinones, F.J. Cazorla, G. Bernat, and M. Valero. Hardware support for WCET analysis of hard real-time multicore systems. In *ISCA*, 2009.

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

11 Wilhelm R. et al. The worst-case execution-time problem overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7:1–53, May 2008.