# A Case Study for a PTA-friendly Processor

Leonidas Kosmidis[1], Luca Santinelli[2], Michael Houston[3], Liliana Cucu[2],
Eduardo Quinones[1], Jaume Abella[1], Tullio Vardanega[4], Francisco J. Cazorla[1,5]
[1]Barcelona Supercomputing Center [2]INRIA [3]Rapita Systems
[4]University of Padua [5]Spanish National Research Council (IIIA-CSIC)

***Abstract*—This document provides a case study for a PTA-friendly processor where a number of complex processor resources have been adapted to enable independence and identical distribution at the level of instruction.**

## I. Designing the architecture of a PTA-friendly processor

We use a 4-stage pipelined core architecture. The four stages operate as follows: Instructions are fetched in-order from the instruction cache (I$) into the processor. The ITLB is accessed in parallel to translate pages; Instructions are decoded in one cycle. Decoded instructions are issued in-order to the execute stage; Instructions are executed in one cycle, except for memory operations. Read operations access the data cache (D$) during this cycle blocking the execution of further instructions until data are fetched. The DTLB is accessed in parallel with the D$; The instructions commit in one cycle. Write operations update the data cache during this cycle.

The pipeline structure is quite similar to that of the LEON4 processor [2] deployed by the European Space Agency.

I$ and D$ are 16KB in size, 8-way set-associative, with 16 bytes/line. The D$ and I$ use the random placement and replacement policies presented in [6]. The D$ is write-through, so that all store instructions are propagated to memory. The D$ is write-allocate, hence data are fetched on a store miss. ITLB and DTLB are 8-entry fully-associative random-replacement, with 1KB page size. The hit and miss latencies are 1 and 80 cycles respectively. The TLB are accessed in parallel with the caches so that instructions are stalled in the corresponding stage until both the cache *and* the TLB can serve the request.

We use benchmarks from the EEMBC autobench benchmark suite [7] for the evaluation of our proposal.

## II. Checking the i.i.d. hypothesis

In our architecture, there are two main sources of execution time variability, TLB and caches, that introduce random events in the execution of a program. Fully-associative TLB lead to a probability of hit per access $phit_{TLB} = (\frac{N-1}{N})^k$ where $k$ is the reuse distance of the access. In the case of a random placement and random replacement set-associative cache with $S$ sets and $W$ ways, starting from an empty cache state and given the sequence $[A_i, B_1, ..., B_k, A_j]$, where $A_i$ and $A_j$ correspond to accesses to the same cache line and no $B_l$ (where $1 \leq l \leq k$) accesses cache line $A$ (also different from cache line $A$), the probability of $A_j$ to miss in the cache is [6]:

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

As it can be seen, the higher the reuse distance ($k$), the higher the probability of other accesses to evict the cache line, as well as the miss probability. Equation 1 is only fully-accurate when each $Bj$ accesses a different line and none of them accesses the same cache line accessed by $Ai$ and $Aj$, subject to an initial state with an empty cache. However, this is irrelevant for MBPTA, since what really matters is hit/miss to have a probability rather than the particular value of that probability. As long as the ETP exists at the level of dynamic instruction (and it does with time-randomised caches) MBPTA can be applied [1].

We differentiate two types of instructions core (e.g. add, div, mult) and memory (e.g. load and store). Core operations take a variable latency depending on whether they hit in the instruction cache and instruction TLB resources, which are composed in parallel leading to what we call the ETP of the front-end (fend): $ETP_{fend} = (ETP_{I\$} \otimes_p ETP_{ITLB}) \otimes_s ETP_{IMEM}$, where $ETP_{IMEM}$ is the ETP in accessing the instruction memory. Access to the instruction memory only happens in case of miss to the I$ or ITLB. $ETP_{exec}$ is a fixed latency for core operations, e.g. $\{4\}\{1.0\}$. $ETP_{buffer}$ is the delay introduced by the buffers in between the different stages.

$$ETP_{core} = ETP_{fend} \otimes_s ETP_{exec} \otimes_s ETP_{buffer} \quad (2)$$

Memory operations have the same front-end ETP than core operations. The backend latency depends on the time of the datapath (dpath) composed by the data cache and the data TLB, which are accessed in parallel and the latency to memory: $ETP_{dpath} = (ETP_{D\$} \otimes_p ETP_{DTLB}) \otimes_s ETP_{DMEM}$. With this the ETP for memory operations is as follows:

$$ETP_{mem} = ETP_{fend} \otimes_s ETP_{dpath} \otimes_s ETP_{buffer} \quad (3)$$

As shown in [1], the existence of the ETP for each instruction ensures that execution times for the entire program fulfill i.i.d. properties. We contrast this empirically by using proper statistical i.i.d. tests applied on the execution times coming from running EEMBC benchmarks on our processor architecture.

For independence we use the Wald-Wolfowitz (WW) test [3] and for identical distribution hypothesis we use the Kolmogorov-Smirnov (KS) goodness-of-fit test [5]. We use a 5% significance level (a typical value for this type of tests),

TABLE I: Independence and identical distribution tests results (2nd and 3rd columns), and average execution time and pWCET bounds of the complex PTA-friendly processor against an equivalent non-PTA-friendly processor (4th and 5th columns)

| Benchmarks | Statistical tests | | Timing analysis results | |
|---|---|---|---|---|
| | Inde-pence | Identical distribution | Average Exec. Time | pWCET $10^{-16}$ |
| **a2time** | 0.949 | 0.387 | 1.04 | 1.13 |
| **aifirf** | 0.380 | 0.791 | 1.09 | 1.12 |
| **cacheb** | 0.063 | 0.668 | 1.14 | 1.45 |
| **canrdr** | 0.126 | 0.529 | 1.01 | 1.04 |
| **puwmod** | 0.253 | 0.993 | 1.00 | 1.01 |
| **rspeed** | 0.635 | 0.628 | 1.00 | 1.04 |
| **tblook** | 0.127 | 0.252 | 0.87 | 0.94 |
| **ttsprk** | 0.696 | 0.187 | 1.13 | 1.25 |

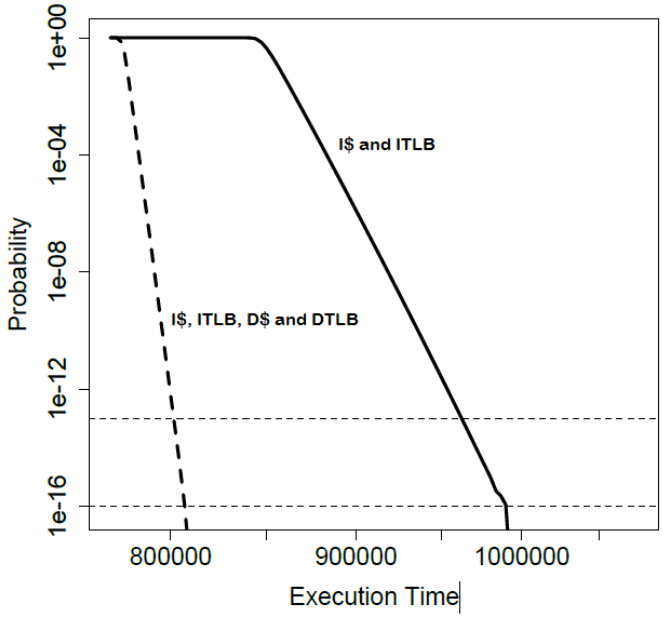


Fig. 1: pWCET estimates for *puwmod* under different setups

which means that absolute values obtained with the WW test must be below 1.96 to prove independence and for the KS test, the outcome provided by the test should be above the threshold (0.05) to indicate identical distribution.

For each such benchmark on the example processor architecture less than 1,000 runs were needed, in line with previous experiences [4], [6]. Note that running 1,000 times applications whose typical execution time is in the order of few milliseconds (as needed in CRTES) implies that pWCET estimates can be obtained in just few seconds. Under the header 'Statistical tests' in Table I we show the results of both tests for all benchmarks. We observe that both tests are passed in all cases, which confirms that our architecture provides the i.i.d requirements needed for PTA.

## III. pWCET

In this section we show the type of probabilistic WCET estimates that can be obtained in our architecture with the method presented in [4]. The solid line in Figure 1 shows the pWCET estimates for *puwmod* benchmark in our baseline architecture from which we have removed the D$ and DTLB. The X-axis shows the execution time and the Y-axis the probability of exceed it. We consider exceedance probabilities of $10^{-13}$ and $10^{-16}$ per run, which fall well within safety automotive and avionics standard regulations for acceptable probabilities of failure. We observe that increasing the exceedance probability does not translate into high increases in the WCET estimates, quite the opposite the pWCET curves have a significant slope. In general, the larger the number of random events (e.g., the number of cache accesses), the lower the probability of abrupt performance variations is. Thus, execution time variation is low and the pWCET curve is sharp.

Our processor architecture provides the requirements needed for PTA to benefit from high performance hardware features. To illustrate this phenomenon, the dotted line in Figure 1 shows the pWCET estimate we obtain if we add the D$ and DTLB in our architecture. We observe that the pWCET estimates are smaller when using the D$ and DTLB (around 20% in our range of probabilities of interest).

To the best of our knowledge complex architectures including caches, TLB, pipelines and buffers have not been unrestrictedly used in the context of static timing analysis, whose WCET estimates rapidly degrade as soon as the address of accesses to those resources cannot be determined at analysis time. Nor can be analysed with MBTA in a trustworthy manner since, for instance, small changes in the way memory objects are allocated in memory may lead to abrupt changes in execution time, which is hard to capture with MBTA techniques.

## IV. PTA-friendly architectures performance

Under the header 'Timing analysis results' in Table I we show average execution time and probabilistic WCET (pWCET) estimates for an exceedance threshold of $10^{-16}$ per run using our complex architecture. Results are normalised with respect to the performance obtained with an analogous architecture that implements modulo placement LRU replacement caches and TLB instead of random placement and replacement. As shown, average execution time degrades only by 4% when using the PTA-friendly architecture. Notably, pWCET estimates are, on average, only 12% higher than the average performance obtained for a processor architecture implementing LRU as the replacement policy for all caches, thus proving that PTA-friendly designs are competitive.


## Acknowledgements

The research leading to these results has been funded by the European Community's Seventh Framework Programme [FP7/2007-2013] under the PROARTIS Project (grant agreement 249100) and the PROXIMA Project (grant agreement 611085).


## References


[1] J. Abella et al. Measurement-based probabilistic timing analysis and i.i.d property. White Paper. 2013. http://www.proartis-project.eu/publications/MBPTA-white-paper.

[2] Aeroflex Gaisler. *Quad Core LEON4 SPARC V8 Processor - LEON4-NGMP-DRAFT - Data Sheet and User's Manual*, 2011.

[3] J.V. Bradley. *Distribution-Free Statistical Tests*. Prentice-Hall, 1968.

[4] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzeti, E. Quinones, and F. Cazorla. Measurement-based probabilistic timing analysis for multi-path programs. In *ECRTS*, pages 91–101. IEEE, July 2012.

[5] M.H. DeGroot and M.J. Schervish. *Probability and statistics*. Addison-Wesley, Reading MA., 2002.
[6] Leonidas Kosmidis, Jaume Abella, Eduardo Quinones, and Francisco J. Cazorla. A cache design for probabilistically analysable real-time systems. In *DATE*, 2013.
[7] J. Poovey. *Characterization of the EEMBC Benchmark Suite*. North Carolina State University, 2007.