HRM: Merging Hardware Event Monitors for Improved Timing Analysis of Complex MPSoCs

Sergi Vilardell
Barcelona Supercomputing Center
Universitat Politècnica Catalunya
sergi.vilardell@bsc.es

Isabel Serra
Centre de Recerca Matemàtica
Barcelona Supercomputing Center
isabel.serra@bsc.es

Roberto Santalla, Enrico Mezzetti,
Jaume Abella and Francisco J. Cazorla
Barcelona Supercomputing Center
name.surname@bsc.es

***Abstract*—The Performance Monitoring Unit (PMU) in MPSoCs is at the heart of the latest measurement-based timing analysis techniques in Critical Embedded Systems. In particular, hardware event monitors (HEMs) in the PMU are used as building blocks in the process of budgeting and verifying software timing by tracking and controlling access counts to shared resources. While the number of HEMs in current MPSoCs reaches hundreds, they are read via Performance Monitoring Counters whose number is usually limited to 4-8, thus requiring multiple runs of each experiment in order to collect all desired HEMs. Despite the effort of engineers in controlling the execution conditions of each experiment, the complexity of current MPSoCs makes it arguably impossible to completely remove the noise affecting each run. As a result, HEMs read in different runs are subject to different variability, and hence, those HEMs captured in different runs cannot be 'blindly' merged. In this work, we focus on the NXP T2080 platform where we observed up to 59% variability across different runs of the same experiment for some relevant HEMs (e.g. processor cycles). We develop a HEM reading and merging (HRM) approach to join reliably HEMs across different runs as a fundamental element of any measurement-based timing budgeting and verification technique. Our method builds on order statistics and the selection of an anchor HEM read in all runs to derive the most plausible combination of HEM readings that keep the distribution of each HEM and their relationship with the anchor HEM intact.**

I. Introduction

The complexity of processors in critical embedded systems (CES) continues to increase, with academic and industrial efforts devoted to analyze the use of multicores as the baseline computing solution in future CES. Multicores – and multiprocessor system-on-chip (MPSoC) solutions in general – provide the increasing computing performance needs in CES domains like automotive [1] and avionics [2]. This is, in turn, motivated by the increasing computing requirements in autonomous CES that manage huge amounts of data, e.g. coming from radar, lidar, and cameras; and the implementation of compute-intensive AI algorithms [3]. The other side of the coin is that multicores complicate software timing analysis due to the inherent complexity of cutting-edge hardware functionalities and the difficulties in capturing the contention in the access to hardware shared resources, which causes tasks to affect each others' timing behavior.

Consolidated timing analysis approaches are challenged by the inherent complexity of multicore computing solutions [4], [5] that are increasingly adopted in the CES domain [1], [2]. The complexity of analysing such platforms principally emanates from the implications of multicore execution on the increasingly richer functionalities that CES are required to provide. This has led to a significant interest in providing industrially-amenable solutions to master contention and the entailed multicore interference. Preventing or controlling contention between concurrently-running tasks has been considered as a promising direction with some approaches building on full segregation of accesses to the different blocks of memory-like resources [6], [7], including the (i) banks of shared on-chip caches and the (ii) banks/ranks in a DDR memory system [8], [9], [10], with solutions combining (i) and (ii) [11]. Other works propose changes to the application to precisely split its execution into memory and computation phases to facilitate explicit scheduling of task phases in a way to avoid contention [12], [13]. These approaches, while being embraced in industrial quality solutions [14], are not always applicable in practice, due to hardware characteristics or constraints on the applications semantics. In all these cases, interference can still arise in shared buses or shared buffers, tables and queues in the cache [15], and whenever altering applications' semantics is not an option due to verification and validation (V&V) costs. In the NXP T2080, considered for adoption by the avionics industry [16], the number of shared components where interference can arise is overwhelming. Just in the L2 cache, we find the back invalidate buffer, reload table, reload fold queue, castout buffer, write data buffer, reload data buffer, and the snoop queue.

In any case, regardless of the specific scenario, an analysis approach is required to provide evidence that contention is actually avoided or mitigated (i.e. its impact can be bounded). Advanced measurement-based timing analysis approaches building on a variably complex combination of software and hardware profiling [17], [18] are being considered as a promising analysis solution for functionally-rich and complex multicore platforms. Measurement-based approaches appear particularly appealing from an industrial (V&V) standpoint [19]. In this view, the Performance Monitoring Unit (PMU) provides the necessary entry point for retrieving the information required by the analysis. In fact, PMUs are becoming instrumental for software timing budgeting and V&V.

As a first example, it has been shown that existing signals in the AMBA AHB bus provide the required information to capture the contention tasks generate each other on the bus. This information is sent to the PMU to be stored in hardware event monitors (HEMs) that are made accessible to the software to track and control contention [20].

Another example is the event quota budgeting, monitoring, and enforcement mechanisms [21], [22], [23], [24] to enforce budgets on task resources utilization by means of HEMs. By controlling task's activities via the HEMs offered by processor's PMUs, the system software can suspend task's execution when their assigned budget is exhausted.

Empirical approaches building on the evidence collected from HEMs, for example, are at the basis of successful certification arguments for CES in the avionics domain [16]. Overall, the PMU, and HEMs in particular, are at the heart of modern solutions to track and control contention in multicores.

Problem Statement. While the number of HEMs in current multicores is in the order of hundreds, they can only be read in small groups of 4-8 via user-visible performance monitoring counters (PMCs). This limitation relates to the hardware cost of routing the HEMs via long wires and multiplexors to access PMCs that can be accessed via software. Hence, several runs are required to read all the HEMs of interest, which are later 'merged' off-line to analyse the program behavior and reason about contention. For instance, to decide whether some tasks can be scheduled concurrently, we need to budget how much each one is expected to access each shared resource, which requires consistent reads of a large number of HEMs.

To make things worse, several runs of the same experiment in an MPSoC can result in inevitable variations in the timing behavior of the program, though its functional behavior is the same. This is due to the impossibility to control the entire hardware and software initial state in each run. In practical terms, this translates into variability in HEM readings (as high as 59% for processor cycles in our target system for relevant HEMs), with no variability observed in instruction count (as analyzed in Section II-C). The engineer is confronted with a set of values (readings) for each HEM, that need to be merged to allow reasoning about multicore contention. Unfortunately, since HEM values from different runs to be merged can be subject to different (large) noise, it is challenging to merge them consistently so that merged HEM vectors – those where all HEMs of interest are included – resemble the values that would have been obtained if they could have been read all of them simultaneously in the same run.

Contribution. The contribution of our work is three-fold:

Analysis of HEM variability. For the NXP T2080 [25], a representative MPSoC in CES and undergoing a certification process of multicore processing in avionics, we make an in-depth statistical analysis of variability in HEM readings. The observed variability is high enough to jeopardize the consolidated practice of 'blindly' merging different HEMs as input for timing analysis. Moreover, the values of those HEMs follow different distributions.

HRM. We introduce a HEM-reading merging (HRM) approach to guide the merging of HEM values subject to different noise. HRM identifies an anchor HEM, and defines groups of HEMs, each group with $PMCs - 1$ HEMs plus the anchor. HRM performs several runs for each group of HEMs, ranks HEM values in each group using order statistics on the anchor HEM, and merges those HEMs with the same rank in different groups. Order statistics are non-parametric and hence, can handle the different distributions of the HEM values observed.

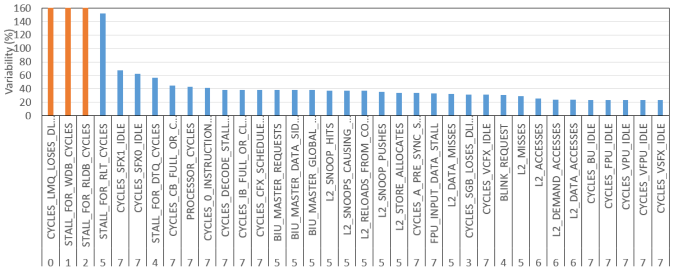

Fig. 1: Observed variability for several HEMs in the T2080

Analysis of the accuracy. Since noise-free HEM values cannot be obtained in general in complex MPSoCs, we evaluate HRM comparing the correlation across HEMs merged by HRM against their correlation when those HEMs are measured in the same run, thus under identical noise. Our results show that HRM captures accurately the correlation between HEMs, as opposed to blindly merging HEMs read in different runs.

The rest of this paper is organized as follows. Section II motivates the need of HEM merging mechanisms and makes an in-depth analysis of HEM variability in the T2080. Section III formalizes the problem addressed in this work Section IV introduces our approach for HEM merging, which we evaluate in Section V. Section VI discusses the most relevant related works. Section VII concludes this work.

II. Motivation

We show that several of the 262 HEMs in the T2080 present significant variation (Section II-A) and follow different distributions (Section II-B). We also dig down into some of the reasons behind the observed variation (Section II-C).

A. HEM variability

On the NXP T2080 [26] we run a four-task workload with each task pinned to one of its e6500 cores. Each task performs integer and floating operations at the core level over several large vectors so that data operated is fetched from main memory, causing frequent misses in all cache levels, and thus exercising several HEMs. For this experiment, as well as the remaining ones throughout this paper, we run on baremetal to remove potential interference coming from the operating system (the specifics of our experimental framework are described in Section V-A). We divide the experiment into several sub-experiments, in each of which we read 6 HEMs (the total number of PMC available in the T2080). Hence, reading all 262 HEMs requires 44 sub-experiments, each of which we repeat 100 times to capture the impact of noise on HEM readings. In all runs we focus on the HEMs for core 0. Each run finishes when the task in core 0 finishes.

Figure 1 shows the maximum relative variability observed for several HEMs, i.e. $var = (max - min)/min$, with bars in the figure sorted from higher to lower. Each bar is tagged (see bottom part of the figure) with the order of magnitude m of the value of each HEM in the experiment. For instance, for $m = 3$, $10^3 \leq hem^i < 10^4$. This information allows assessing the potential impact of the variability on execution time, whose magnitude for this experiments is tens of millions (10^7) of cycles. So is that for the number of committed instructions.

In the NXP T2080, the maximum duration of an event triggered by an instructions can be in the order of hundreds cycles (10^2). Hence, HEMs below 10^4 arguably have low impact on performance. This, of course, is related to events not involving the execution of system software, e.g. a TLB miss, whose impact is not covered by multicore contention timing analysis but instead captured by the system-level timing analysis. We differentiate some cases for our experiments.

- **Relevant high variability**. Some HEMs present high variability while their magnitude is relevant, 10^4-10^7. These HEMs are the focus of our study as they can significantly impact the timing of the application and hence, the bounds that can be derived to it. In this category we find `PROCESSOR_CYCLES` with a variability of 45% from $3.6 \cdot 10^7$ to $5.2 \cdot 10^7$ (in other experiments the variability of this HEM reached 59%). 49 HEMs fall in this category if we set 1% as threshold for low-variability.
- **Irrelevant or low variability**. Other HEMs have low variability in absolute terms, thus having little impact on performance. There are 58 HEMs in this category, including the three on the left of Figure 1 whose variability is over 180% but their value is below 300, hence, insignificant w.r.t. the cycle count. Other HEMs, 5 in total for this experiment, while exhibiting values above 10^4, incurred less than 1% variability, with limited impact on performance.
- **Not exercised**. Finally, other HEMs, 150 in our case, were not exercised by the program under analysis making both the minimum and maximum value be zero. As the set of HEMs exercised can change across different experiments, the particular non-exercised HEMs will like vary. In fact, this is the motivation behind having different benchmarks in the experimental evaluation.

The observed HEM variability does not depend on the particular subset of HEMs that are enable/disabled when collecting observations. Interestingly, the 'low variability' category comprises HEMs presenting no variablity. Those are related to the functional execution of the program capturing the number of completed instructions including SFX, CFX, store, load, stores, taken and non-taken branches. For instance, the total number of executed instructions (`INSTRUCTIONS_CMPLTD`) of the first task is exactly the same in all runs (15,646,749). This leads us to conclude that the observed variability does not come from the software that always performs the same function (e.g. it traverses always the same execution path), and instead the variability is induced by the hardware.

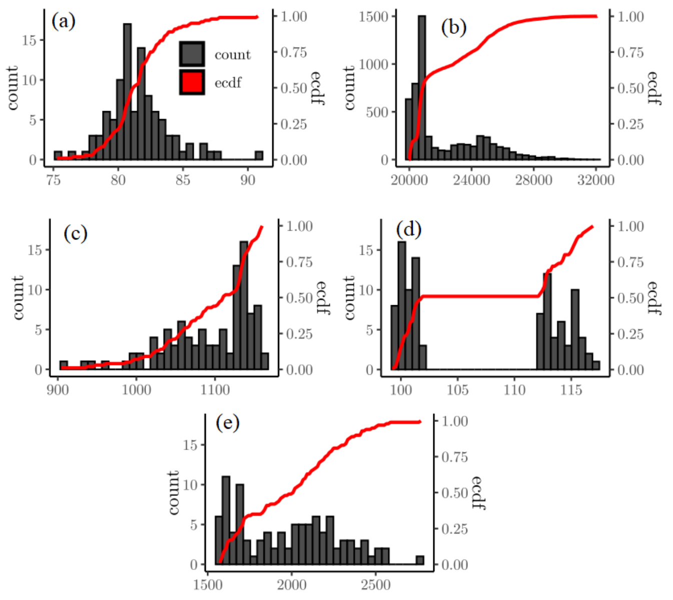

Fig. 2: Histogram and empirical CDF (ECDF) types: (a) Normal, (b) Concave, (c) Convex, (d) Clustered, (e) hard-to-fit.

B. Distribution

Focusing on the *relevant high variability* HEMs, we identified their variable behavior falls into five main classes of distribution. These types are depicted in Figure 2 that shows the histogram (bars) and the cumulative distribution function or CDF (line) of observed values for one HEM in each category for illustrative purposes. The x-axis shows HEM value, the left y-axis the frequency of occurrence for the histogram (for a 500 observations sample), and the right y-axis the fraction of observations for the CDF.

a. **Normal**. HEMs in this category show a symmetric behavior that resembles a normal distribution.
b. **Concave**. The distribution resembles a uniform with leaning towards the smallest values, which gives a concave cumulative distribution function.
c. **Convex**. Distribution with the probability mass concentrated on the highest values of the distribution, giving a convex cumulative distribution function.
d. **Clustered**. HEMs in this category show a clustered behavior around two values or more values. Distributions with more than one clear mode also fall into this category.
e. **Hard to fit**. Finally, some distributions follow no obvious distribution (hard to fit category) apparently characterized by having two modes and a long tail.

Out of the 37 relevant HEMs in our experiment, their distribution is as follows: 2 (5.4%) Normal, 13 (35.1%) concave, 19 (51.4%) convex, 1 (2.7%) clustered, and 2 (5.4%) hard-to-fit. The case of the HEM `PROCESSOR_CYCLES` is particularly relevant for a two-fold reason. It is the main HEM used in timing analysis for making predictions, and it presents a hard to fit distribution (see Figure 2(e)). This HEM presents 59% variability from around $2.0 \cdot 10^7$ to $3.2 \cdot 10^7$.

C. Reasons Behind the Observed Variability

The T2080 implements a complex architecture with an aggressive core (the e6500), so some form of hardware-induced HEM variability is therefore expected. We have observed that the HEMs with relevant high variability capture the activity in a wide range of hardware units, from the (on-core) integer issue queue to the internal queues of the L2 cache. High variability can be due to the complex nature of the T2080 and its sources of multicore interference: specific hardware scheduling choices in the multiple shared queues and buffers in the core-to-L2 interconnect, internal to the L2, the CoreNet Coherence Fabric (CCF), and the memory controller, may lead to variable latencies for specific requests. Specific and controlled execution scenarios allow narrowing down the sources of execution time variability. As an example, we have performed some bare-metal experiments where all cores hit L2 cache sustainedly, with a task τ overlapping its full execution with the others. The intent is that interference occurs solely in the L2 cache. Variability observed across executions (up to more than 40%) could be attributed to minor initial processor state differences causing slight time shifts between L2 accesses across runs, and leading to cascade effects in L2 queues. In general, however, the limited information about the internal functioning of some of these resources, e.g. CCF, simply prevents identifying some of the reasons behind the observed variability. Also, as the programs used in this study typically perform the same activities repeatedly, contention for requests of a given core can stay repeatedly low or repeatedly high, leading to cumulatively high variability. Further, such systematic patterns may inadvertently switch from low to high contention scenarios (or vice versa) due to several reasons, such as the effects of loop control instructions in a program, which might alter systematic behavior inside the loop, as well as the impact of DRAM refresh operations, just to name some examples.

Authors in [27] perform a hardware analysis of several Intel architectures and formulate several hypotheses on the reasons behind various forms of under and over counting affecting some HEMs (retired instructions, branches, load/stores, floating point, etc). Extending this to modern MPSoCs confronts with the inclusion of large hardware IP blocks with limited description and the increasing number of HEMs monitoring events highly sensitive to such variability. Also, note that knowing the reasons behind such variability would help assessing whether the device allows some configurations under which the variability reduces. However, if the behavior causing the variability is intrinsic to the complexity/functioning of the device, a solution like HRM is still needed – whether or not the root can be explained.

D. Disproportion between the number of HEMs and PMCs

As the complexity of MPSoCs in modern CES platforms in domains like avionics and automotive continues to increase, so will do the number of HEMs. In fact, current MPSoCs, already comprise hundreds of HEMs. For instance, the ARMv8(-A) architecture defines over 280 micro-architectural events [28]. This architecture is implemented by a set of processors such as the A53 (used by the NXP LayerScape family and the Xilinx ZynqUltraScale MPSoC), the A57, the A72, and NVIDIA's Carmel processor. Each processor implements a subset of those HEMs. For instance A53 implements 63, while more modern A57/A72 implement 92/85 respectively. We see a similar increasing trend in the NXP eXXX family with 180 HEMs in the e500mc and 262 in the e6500.

Despite the increase in number and specialization of HEMs, their observability and accessibility in reference CES platforms is typically constrained by the availability of a relatively (but consistently) lower number of PMCs. The latter represent, in fact, the most natural way of making HEM information available to the user. The number of PMCs available in modern MPSoCs typically ranges between 4 and 8 per core, which inherently clashes with the number of HEMs an analysis would need to track. In the case of Arm A53/A57/A71 cores, the number of PMCs is 6, which also matches the number of PMCs in NXP cores e500mc/e6500. To further constrain the observability of HEMs, some platforms also enforce limitations on how PMCs can be configured and mapped to HEMs. In the Infineon AURIX TriCore family of microcontrollers, reference computing solution in the automotive domain, the number of available PMCs is limited to 3 per core, as compared to 14 HEMs (2 fixed and 12 configurable), but only specific groups of HEMs can be tracked at the same time [29]. For example, it is not possible to track both instruction and data cache related HEMs within the same experiment. Overall, the clear unbalance between the large number of HEMs and available PMCs calls for approaches that reliably merge HEM readings across different experiments.

It is also worth mentioning that MPSoCs also include an increasing number of complex shared resources. This will naturally result into more HEMs tracked by timing analysis techniques to capture the effect of contention. While the particular multicore timing analysis solution determines the number of HEMs to track, as an illustrative example we show that getting a 'snapshot' of the usage of the L2 cache made by a task in the T2080, requires tracking dozens of HEMs. The resources involved are the instruction cache, data cache and L2 MMU, whose misses go to L2; the core-cluster interface (CCI) that connects cores to the L2, the L2 itself, and the bus interface unit (BIU) that connects the L2 to the CCF. A (simple) analysis can focus on retrieving access counts to these resources, without analyzing their effect propagation to the core, which would require tracking more HEMs. Such analysis might need to track 53 HEMs: instruction cache misses (3), data cache misses (4), L2 MMU misses (3), CCI accesses (5), L2 activity (32), and BIU accesses (6).

III. Problem Formalization

We are interested in collecting the values of a set of relevant HEMs, $\mathcal{H} \ni \{h^1, h^2, \cdots, h^{nh}\}$, whilst a given program executes on the target platform in response to a given input (the main terms used in this paper are listed in Table I).

TABLE I: Main terms used in this work.

Term	Definition
nh, np, nb, nr	number of HEMs, PMCs, sub-experiments, and runs.
h^i	HEM with id i
b_j	j-th subexperiment
$r_{j,k}$	run k of a given subexperiment b_j
$m^i_{j,k}$	measured value for h^i in run $r_{j,k}$
M^i_j	set of measured values for h^i in b_j
$v^i_{j,k}$	range of variation of measured value $m^i_{j,k}$
h^a	anchor HEM
$sr_{j,l}$	Run of b_j with the l^{th} lowest value of h^a
SR_l	Concatenation of $sr_{j,l}$ for all $\{b_j\}_{j=1,\cdots,nb}$
SM_l	HEM readings in $sr_{j,l}$ for all $\{b_j\}_{j=1,\cdots,nb}$

In an ideal scenario, all nh HEMs are collected at once on a single program execution without incurring the uncontrolled (platform or system level) jitter or variability that may arise across executions. Under such favorable conditions, we obtain a set of measurements (values) for each HEM in $h^i \in \{\mathcal{H}\}$ that cumulatively capture the activity performed by the program. This is referred to as Scenario 1 in Figure 3, in which the row shows the single execution, columns the HEMs and the cell their respective values.

In a more realistic scenario, program executions on the target platform are subject to *noise* so that in each execution the measured values for a given HEM h^i can potentially vary. Note that we use the term 'noise' to generically refer to the varying execution conditions across experiments, either due to different initial hardware and system software state (in this respect, our experiments are executed baremetal reducing the variability due to system software). We are not after quantifying such noise, but we just recognize that it is in general uncontrollable, beyond the measures we take in order to reduce it as shown in Section V-A. To capture the impact of noise, several runs of each experiment need to be performed. The noise of the different runs is represented as different levels of grey in Figure 3 (Scenario 2). In this scenario, noise can occur but at least all HEMs can be read at once, so all HEMs in each run are exposed to the same noise. This makes possible to reason about their relationships for statistical inference.

In general, however, it is not possible to read all nh HEMs at once in a single execution as the number of HEMs that can be tracked simultaneously is determined by the number of available PMCs. Assuming our platform support np configurable PMCs[1] typically comparatively small with respect to the number of supported HEMs, with $np << nh$. For this reason, HEMs are necessarily collected in groups of at most np elements. Hence, to measure all HEMs for a given program we must perform a set of at least $nb \geq \lceil nh/np \rceil$ sub-experiment $(b_1, \cdots, b_j, \cdots, b_{nb})$, each capturing the values of at most np distinct HEMs and cumulatively covering all HEMs.

[1]Without lack of generality, we assume there are no constraints on which specific HEM can be read from each PMC. Some processors exhibit such constraints, due to hierarchy of multiplexors to route HEM readings to a specific PMC. This scenario would just restrict which HEMs can be read in the same run, but would not affect HRM, as some HEMs (e.g. processor cycles) can be read along with any other group of HEMs

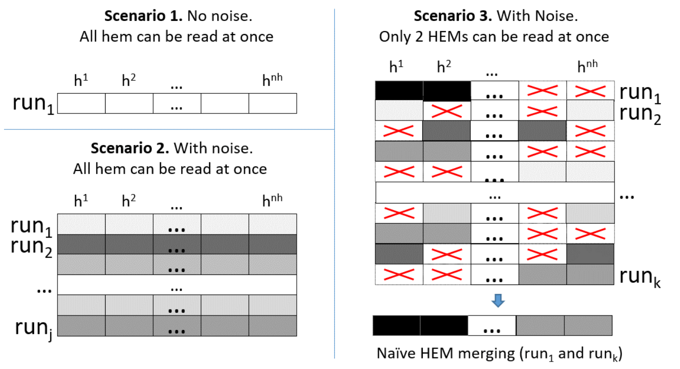
Scenario 1. No noise.
All hem can be read at once
h1
h2
...
hnh
run1
Scenario 2. With noise.
All hem can be read at once
run1
run2
runj
Scenario 3. With Noise.
Only 2 HEMs can be read at once
runk
Naïve HEM merging (run1 and runk)

Fig. 3: Scenarios in HEM reading

To capture the variability in measured values, several runs of the same sub-experiment b_j are carried out, see Scenario 3 in Figure 3, with crosses showing the HEMs not read in a given run. In this case, we assume only 2 HEMs can be read in each run. Also, as shown at the bottom of Scenario 3, naively merging HEMs (from the first run and kth run in this case) results in merging HEMs values obtained under different noise levels, potentially resulting in inconsistent values that cannot be reliably used.

In this paper, we address the challenge of merging the readings (measurements) for all $h^i \in \mathcal{H}$, each one measured several times in a different sub-experiment (Scenario 3) to obtain noise-consistent measurements for all HEMS (Scenario 2), preserving their relationships with execution time to favor timing analysis. Note that, noise-free HEM values (Scenario 1) are arguably hard to achieve, if at all possible, in MPSoCs. In particular, we aim at obtaining vectors with values for all HEMs under *similar* noise, as if all of them could have been read simultaneously in every single run.

IV. HRM: A TECHNIQUE TO MERGE HEMS

Table II introduces an example with the main inputs and outputs to be generated by any HEM merging approach. In particular, it shows the measurements made when the number of HEMs is $nh = 12$ and the number of PMCs is $np = 3$, hence being required $nb = 4$ sub-experiments. In the example, $nr = 5$ runs are performed per sub-experiment. On the left, it is reported the sub-experiment and run id. In the top part, the HEM id. We use $r_{j,k}$ to refer to the run k of sub-experiment b_j. We refer to measured value of HEM h^i in b_j and run k as $m^i_{j,k}$. In terms of outputs, a HEM merging mechanism must aim at producing a list of nr all-HEM readings (vectors) where each vector includes all HEMs. Each of the nr measurements of each HEM is placed exactly in one of those nr vectors. This is illustrated at the bottom of Table II, where each run of each sub-experiment is merged with another run from each other sub-experiment so that each run is represented exactly once in the merged result. For instance, in the example, the 1st run of the first sub-experiment ($m^1_{1,1}$, $m^2_{1,1}$, $m^3_{1,1}$) is merged with the x'^{th} run of the 4th sub-experiment, and with one run of each other sub-experiment represented as the x^{th} run of the j^{th} sub-experiment.

TABLE II: Example with $nh = 12$, $np = 3$, $nb = 4$, a $nr = 5$.

		h^1	h^2	h^3	$\cdots$	h^i	$\cdots$	h^{10}	h^{11}	h^{12}
b_1	$r_{1,1}$	$m^1_{1,1}$	$m^2_{1,1}$	$m^3_{1,1}$						
	$\vdots$	$\vdots$	$\vdots$	$\vdots$						
	$r_{1,k}$	$m^1_{1,k}$	$m^2_{1,k}$	$m^3_{1,k}$						
	$\vdots$	$\vdots$	$\vdots$	$\vdots$						
	$r_{1,5}$	$m^1_{1,5}$	$m^2_{1,5}$	$m^3_{1,5}$						
					$\ddots$					
b_j	$\vdots$					$\vdots$				
	$r_{j,k}$				$\cdots$	$m^i_{j,k}$	$\cdots$			
	$\vdots$					$\vdots$				
							$\ddots$			
b_4	$r_{4,1}$							$m^{10}_{4,1}$	$m^{11}_{4,1}$	$m^{12}_{4,1}$
	$\vdots$							$\vdots$	$\vdots$	$\vdots$
	$r_{4,k}$							$m^{10}_{4,k}$	$m^{11}_{4,k}$	$m^{12}_{4,k}$
	$\vdots$							$\vdots$	$\vdots$	$\vdots$
	$r_{4,5}$							$m^{10}_{4,5}$	$m^{11}_{4,5}$	$m^{12}_{4,5}$

$\Downarrow$

SM_1	$m^1_{1,1}$	$m^2_{1,1}$	$m^3_{1,1}$	$\cdots$	$m^i_{j,x}$	$\cdots$	$m^{10}_{4,x'}$	$m^{11}_{4,x'}$	$m^{12}_{4,x'}$
		$\vdots$			$\vdots$			$\vdots$	
SM_i	$m^1_{1,k}$	$m^2_{1,k}$	$m^3_{1,k}$	$\cdots$	$m^i_{j,y}$	$\cdots$	$m^{10}_{4,y'}$	$m^{11}_{4,x'}$	$m^{12}_{4,y'}$
		$\vdots$			$\vdots$			$\vdots$	
SM_{nr}	$m^1_{1,5}$	$m^2_{1,5}$	$m^3_{1,5}$	$\cdots$	$m^i_{j,z}$	$\cdots$	$m^{10}_{4,z'}$	$m^{11}_{4,x'}$	$m^{12}_{4,z'}$

A. Approach

Our approach, HRM, builds on non-parametric order statistics, which allows relating random variables based on the order of the sampled values of the variables, regardless of their distributions. In particular, HRM aims at merging the HEM measurements from different sub-experiments in such a way that the noise experienced by the different measurements is as similar as possible. HRM must also allow merging HEMs regardless of the distribution of the data to be merged. Non-parametric order statistics, which resort to the order of data regardless of their distribution, allow relating runs across measurements through the use of an 'anchor' HEM, referred to as h^a, measured in all sub-experiments. HRM derives the relation between HEMs in different sub-experiments via their relation to h^a.

This is illustrated in the left side of Figure 4 that shows how the individual readings of h^i and h^j (m^i and m^j respectively) from different sub-experiments are related to those of the h^a in each sub-experiment referred to as. HRM provides the following properties. First, it preserves the distribution of each individual HEM. It also preserves the joint distribution between each HEM, h^i, and the anchor HEM, h^a (recall that the joint distribution between the HEMs read in the same sub-experiment is maintained). Finally, HRM estimates the most reliable joint distribution across HEMs in different sub-experiments. Next we detail the procedure followed by HRM to provide those properties, followed by the mathematical

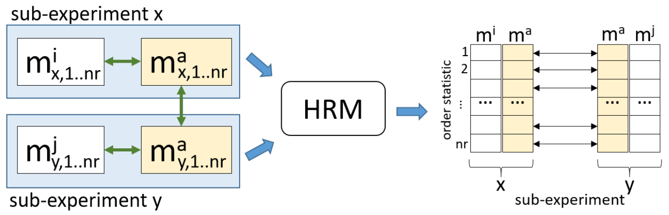

Fig. 4: Introduction to the HRM approach.

foundation of the approach.

B. Procedure

The application process of HRM includes four main steps.

STEP ①. HRM starts by selecting the anchor HEM, h^a, that will be read in all sub-experiments. In each sub-experiment $np - 1$ PMCs are used to read different HEM. That is, from all available np PMC, HRM uses one of them in each sub-experiment b_j for the anchor, and the other $np - 1$ PMCs for other HEM. HRM approximates unobserved HEM relationships via their individual (observed) relationship with h^a. Thus, the selection of h^a is critically important as it determines how effective is HRM to merge HEMs for the problem under study. As the problem at hand relates to timing analysis, we chose h^a to be as relevant as possible to timing. In the case of the T2080, execution time is measured via the HEM `PROCESSOR_CYCLES`, and hence $h^a =$'PROCESSOR_CYCLES'.

STEP ②. After performing nr runs of each sub-experiment, HRM sorts the runs of each sub-experiment by h^a, from lowest to highest. As a result, each element in the sorted list for each sub-experiment will indicate an order statistic, with the k^{th} order statistic of a sample being its k^{th}-lowest value.

Each sub-experiment is characterized by a small fixed set of HEMs, limited by the number of PMCs available, np. Each sub-experiment in $\{b_j\}_{j=1,\cdots,nb}$ is represented by a set of nr runs of dimension np,

$$r_{j,k} : (m^a_{j,k}, m^{(np-1)(j-1)+1}_{j,k}, \cdots, m^{(np-1)(j-1)+(np-1)}_{j,k})$$

The selection of nr should be based on prior knowledge of h^a random variable behavior. Without such knowledge, one must resort to $nr \geq 30$, as this is the minimum size to estimate the main properties of a distribution through the central limit theorem. Runs in each sub-experiment and across them, should be designed to ensure that they are independent and identically distributed to enable the probabilistic reasoning on which HRM builds. To achieve this property, we empty the processor state between runs (see Section V-A). We assess it by performing statistical independence and identical distribution tests (see Section V-C).

STEP ③. Once all sub-experiments are sorted based on h^a, we merge the different sub-experiments so that the k^{th} measurement in the list for h^i in a given sub-experiment is merged with the k^{th} measurement of h^j in another sub-experiment. Naturally, HEM measurements in the same run of the same sub-experiment remain at exactly the same position in the sorted list, so they remain together upon merging.

Let $sr_{j,l}$ be the run of sub-experiment b_j with the l^{th} lowest value of h^a. $sr_{j,l}$ is defined as $sr_{j,l} = r_{j,k}$ where $m^a_{j,k}$ is the l-lowest value in the set $\{m^a_{j,k}\}_{k=1,\cdots,nr}$. Finally, the concatenation produces a vector with completed representation of HEMs $SR_l = (sr_{1,l}, \cdots, sr_{j,l}, \cdots, sr_{nb,l})$ for each $l = 1, \cdots, nr$. Since the l-lowest value of $m^a_{j,k}$ is well-defined, we can assume that for each $j = 1, \cdots, nb$, $m^a_{j,k} \leq m^a_{j,k+1}$ for all $k = 1, \cdots, (nr-1)$. Therefore, $m^a_{j,k}$ is the $(k : nr)$-order statistic of the sample of size nr, $\{m^a_{j,k}\}_{k=1,\cdots,nr}$. HRM merges the values read in the same ordered run across all sub-experiments, referred to as SM_l. For each SM_l, HRM produces one reading for each HEM and nb readings for h^a.

STEP ④. After merging, we compute the summarized order statistics for h^a. In particular, we compute the quantiles of the distribution of all values of h^a across all sub-experiments so that we obtain exactly nr quantiles, i.e. one for each row of our merged list of HEM values. The resulting array yields $\hat{m}^a = \text{quantile}(0, \cdots, k/(nr-1), \cdots, 1)$, where $k = 0, \cdots, (nr-1)$. HRM estimates the nr equal spaced quantiles of h^a using the sample of size $(nr \cdot nb)$ obtained from joining all h^a values.

C. Quantile Estimation

Several methods for quantile estimation can be considered. Let $\{x_{(k)}\}_{k=1,\cdots,n}$ be an ordered sample of size n. In general, a method for quantile estimation corresponds to weighted averages of consecutive order statistics. Given fixed values for a function γ and a constant m, the p-quantile is defined by $q(p) = (1-\gamma(j,m))x_{(j)} + \gamma(j,m)x_{(j+1)}$, where $(j-m)/n \leq p < (j-m+1)/n$, $x_{(j)}$ is the $(j : n)-$order statistic. We consider a continuous representation of quantile estimation with $\gamma(j,m) = p \cdot n + m - j$ and $m = 1-p$, which is equivalent to do linear interpolation between the points $\{(p_k, x_{(k)})\}$ where p_k attempts to estimate the mode of $F(x_{(k)})$. Then $q(p)$ is a continuous function of p and $p(k) = (k-1)/(n-1)$. We refer the interested reader to [30] for a review of programming quantile estimation.

D. Correlation Boundary

HRM produces a solution that preserves the observed information and reliably builds unobserved information by preserving joint distributions. That is, HRM preserves the correlation across HEMs. In particular, HRM describes the relationship between the expected value of a target HEM, the anchor h^a, and the values observed for a different HEM h^i. The relationship across different HEMs, namely h^i and h^j, not observed together, is built therefore through h^a. HRM builds such relationship by estimating the covariance matrix across all HEMs. The covariance matrix can be used because each h^i is a random variable with at least 4 *finite moments*. In our case, each HEM has infinite finite moments since all HEM values are bounded, i.e. they count finite events per cycle during a finite interval, since the measurement starts until the measured value is collected. Therefore, each HEM value is a bounded number, thus guaranteeing the existence of infinite finite moments. By being random variables with finite moments, we can describe the relationship across expected values of HEMs through a multivariate normal distribution based on the central limit theorem, which ultimately ensures the existence of the covariance matrix that characterizes the relationship between the expected values of HEMs asymptotically.

In particular, correlation across HEMs is based on Pearson correlation coefficient [31]. It can be obtained via a Least-Squares fit, where a value 1 represents a perfect positive relationship, -1 a perfect negative relationship, and 0 the absence of any relationship across variables. Let X and Y be random variables, and denote by $cor(X,Y)$ the Pearson correlation, obtained as $\frac{\text{cov}(X,Y)}{\sigma_x \sigma_y}$, where σ and cov describe the variances and covariance, respectively.

Lemma. Let (Y, X_1, X_2) be a random vector with multivariate standardized normal distribution. Then, the correlation between X_1 and X_2 is in the interval

$$\rho_1\rho_2 \pm \sqrt{1-\rho_1^2}\sqrt{1-\rho_2^2},$$

where $\rho_i = cor(Y, X_i)$ for $i = 1, 2$.

Proof. Let Σ be the covariance matrix the joint distribution between the random variables, Y, X_1 and X_2. Σ can be described as the correlation matrix

$$\begin{pmatrix} 1 & \rho_1 & \rho_2 \\ \rho_1 & 1 & \rho \\ \rho_2 & \rho & 1 \end{pmatrix} \tag{1}$$

where ρ cannot be arbitrarily set between $[-1, 1]$, since the matrix must be positive semidefinite. A Hermitian matrix is positive semidefinite if and only if all principal minors are non-negative. Building on Silvester's criterium, only the minors defined by submatrices starting from the upper left corner need being checked. The 2-by-2 submatrix, $\left(\begin{smallmatrix} 1 & \rho_1 \\ \rho_1 & 1 \end{smallmatrix}\right)$, is trivial, and the 3-by-3 matrix produces the result to prove.

Note that, by operating the result of the multivariate standardized normal distribution with the corresponding μ and σ of the random variables of the HEMs whose joint distribution we are studying, we can directly obtain the result for the multivariate non-standardized normal distribution. Moreover, since the random variables studied (the HEMs) have 4 finite moments (in fact they have infinite moments), and based on the central limit theorem, the Lemma guarantees that observed values converge asymptotically to the expected values.

Building on the Lemma, we can prove that HRM guarantees the three properties described in Section IV-A.

Theorem Let H be the joint distribution of all HEMs, and assume the set of ordered sub-experiments as shown in the example in Table II. Let be $\{\hat{m}^a_k\}_{k=1,\cdots,nr}$ the set of nr equal spaced quantiles of h^a from the sample $\{m^a_{j,k}\}_{j,k}$ of size $(nb, \cdot, nr)$. Consider the complete (merged) vector with all HEMs defined as:

$$(\hat{m}^a_k, m^1_{1,k}, \cdots, m^{(np-1)}_{1,k}, \cdots$$
$$m^{(np-1)(j-1)+1}_{j,k}, \cdots, m^{(np-1)(j-1)+(np-1)}_{j,k}, \cdots$$
$$m^{(np-1)(nb-1)+1}_{nb,k}, \cdots, m^{(np-1)(nb-1)+(np-1)}_{nb,k})$$

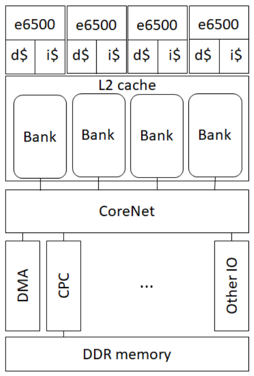

Fig. 5: Block diagram of the T2080

Name	ID
CYCLES_LSU_SCHE_STALLED	1
CYCLES_LSU_ISSUE_STALLED	2
BLINK_REQUEST	3
L2_MISSES	4
L2_DEMAND_ACCESSES	5
L2_ACCESSES	6
L2_STORE_ALLOCATES	7
L2_DATA_MISSES	8
L2_RELOADS_FROM_CORENET	9
L2_SNOOP_HITS	10
L2_SNOOP_PUSHES	11
STALL_FOR_RLT_CYCLES	12
STALL_FOR_WDB_CYCLES	13
BIU_MASTER_REQUESTS	14
BIU_GLOBAL_REQUESTS	15

Fig. 6: HEMs with observed relevant variability.

for each $k = 1, \cdots, nr$. Then, the empirical joint distribution described by the complete vectors complies the (formalized) properties of HRM:

- *Property 1*. Preserves the marginal distribution of H for all HEMs.
- *Property 2*. Preserves the joint distribution across HEMs in the same sub-experiment.
- *Property 3*. Estimates, with minimum error on correlation, the joint distribution between HEMs in different sub-experiments.

Proof. *Property 1*: The marginal distribution of all HEMs but h^a is preserved, since no modifications are produced in the observed values of those HEMs – they are just sorted. Only h^a is modified, since it is replaced by the order statistics. As its distribution has infinite moments, replacing h^a by its order statistics of a larger sample, leads to a higher amount of information (i.e. $nb \cdot nr$ values instead of nr), improving the sample and hence, preserving the marginal distribution of h^a.

Property 2: The re-ordering procedure preserves measurements of different HEMs in the same sub-experiment together. Therefore, their joint distribution is preserved identical.

Property 3: Regarding the joint distribution of HEMs in different sub-experiments, HRM estimates such joint distribution for each pair of HEMs. Note that, since those pairs of HEMs are never observed in the same sub-experiment, data collected provides no information about their joint distribution. The only relation across sub-experiments is had through h^a, which is observed in all of them, so the joint distribution to be estimated needs to preserve this common relation. Based on the Lemma, such relation is preserved if the estimated correlation for describing the real joint distribution of two HEMs in different sub-experiments is in the interval $\rho_1\rho_2 \pm \sqrt{1-\rho_1^2}\sqrt{1-\rho_2^2}$, where ρ_1 and ρ_2 are the correlation between each of those two HEMs and h^a. Since there is no additional information about the actual correlation between those two HEM, any value in the interval is equally probable. Thus, the correlation value proposed by HRM is $\rho_1\rho_2$, since this is the value that minimizes the absolute error w.r.t. the real value.

TABLE III: Workloads on the T2080 for validation purposes

	Core0	Core1	Core2	Core3
W1	IMUL_UL2	FADD_MEM	FMUL_MEM	LMUL_MEM
W2	LADD_DL1	LMUL_UL2	FADD_UL2	FMUL_UL2
W3	LMUL_MEM	LMUL_UL2	FADD_UL2	DMUL_UL2
W4	LADD_UL2	FADD_MEM	FMUL_MEM	LADD_MEM
W5	FADD_MEM	FMUL_MEM	LADD_MEM	LMUL_MEM
W6	FADD_UL2	FMUL_UL2	LADD_UL2	LDIV_UL2
W7	FADD_UL1	FMUL_UL1	LADD_UL1	LMUL_MEM
W8	FMUL_UL1	FADD_MEM	LMUL_L1	LMUL_MEM
W9	FMUL_MEM	FADD_DL1	FADD_MEM	LMUL_DL1
W10	LADD_MEM	LADD_DL1	LADD_UL2	LADD_MEM
W11	FADD_DL1	FADD_UL2	FADD_MEM	FMUL_UL1
W12	FADD_UL2	LADD_UL2	LDIV_DL1	LMUL_UL2
W13	LADD_UL2	FADD_MEM	FMUL_MEM	LADD_MEM
W14	LADD_UL2	LMUL_UL2	FADD_MEM	FMUL_MEM
W15	FADD_MEM	FMUL_MEM	LADD_DL1	LDIV_DL1
W16	LADD_DL1	LDIV_DL1	FADD_MEM	FMUL_MEM

E. Matrix Completion Techniques

HRM aims at merging *actual observations* rather than filling missing values with synthetic data. The latter, which may be realized with Matrix Completion (MC) methods [32], [33], as discussed later in Section VI, is not appropriate in our case. This is so because MC requires that values in each row and column belong to a different distribution, which is not our case, since each column is a different HEM with its own distribution. As a consequence, the use of MC methods for our problem leads to inadequate value distributions where, for instance, the mean and standard distribution of the synthetic data for all HEMs is extremely different from those for actual observations. For instance, in our experiments, the mean for synthetic data is $\approx 20x$ smaller than that of real data, whereas the standard deviation is between $0.36x$ and $5x$ that of real data.

V. Experimental Evaluation

A. Experimental Setup

We target a NXP T2080 Reference Board [26], [34] increasingly considered in the avionics domain, with Rockwell Collins pursuing the certification of multi-core processing on this board [16]. The T2080 equips 4 e6500 cores (see Figure 5), each comprising private instruction and data cache as well as a private MMU. A second level cache is shared between all the cores. A "CoreNet" coherence fabric provides access to the memory controller as well as other peripherals present in the board. Some features are deactivated in our setup for predictability reasons, such as SMT (Hyperthreading in Intel terminology) and the CoreNet Platform Cache (CPC).

We have run our tests in a bare-metal setup, using the SDK provided by the board manufacturer (NXP) to configure the platform and load images to it through a JTAG debugging interface. In the bare-metal setup, we access PMCs directly without the use of a specific library, e.g. PAPI, to minimize the impact of readings.

In each experiment, we run one benchmark per core. The task in core0 is the reference task on which we perform the analysis. The analysis for the tasks in the other cores would be performed analogously. In each run of every experiment, we collect measurements when the task in core0 finishes its execution. We consider single-path benchmarks to isolate platform-level variability, so that in all runs the number of instructions executed (`INSTRUCTIONS_COMPLETED`) in core0 is exactly the same. Across any two runs of an experiment, we reset the state of caches, TLBs, and Branch Target Buffer. To that end, we execute a micro-benchmark that generates a massive number of misses in all those stateful blocks. While ISA-specific solutions exist that allow obtaining the same effect with specific instructions, we considered the micro-benchmark solution to be more platform agnostic.

In general, programs can have built-in sources of non-determinism (e.g. time- or input-dependent values). Also, they may easily be subject to variability due to minimal variations in the Operating System [27]. In order to reduce these sources of variability, we construct specific test-cases, which also aim at triggering a wide set of HEMs. To that end, we have created different benchmarks comprising different core and cache (memory) patterns. At core level, we create 3 benchmarks using intensively the integer pipeline, using integer (I) and long (L) operands, and the floating point (F) pipeline. We use short latency addition (ADD) operations and long-latency multiplication (MUL) operations. At the cache hierarchy level, benchmarks operate on a vector whose size we vary so most load/store operations hit in the data cache (DL1), the L2 (UL2) or memory (MEM). From these 18 benchmarks $(I, L, F)x(ADD, MUL)x(DL1, UL2, MEM)$, we have generated 16 workloads, as shown in Table III.

B. Validation Methodology

The validation of any HEM merging methodology is complex on real hardware as we do not have the noise-free value for each HEM (h^i), as explained in Section III. This prevents us from directly comparing the estimated value for each HEM with its corresponding noise-free value. Thus, we can only evaluate HRM comparing the correlation for the HEM merged with HRM against the real – measured – correlation.

In order to evaluate estimated and real correlations, first, for each workload, we perform 100 runs for each of the $53 = 261/5$ sub-experiments. Hence, we collect readings for all 262 HEMs with 5 HEMs plus h^a read in each sub-experiment[2], except the last group (sub-experiment) that only includes 1 HEM and the h^a.

We validate HRM for 15 HEMs having high relevant variability, see Figure 6. To that end we on purpose place those HEMs in different groups so that their mutual correlation is not observed in the data used for HRM.

For each of the 120 pairs[3] of HEMs we estimate their correlation $\hat{\rho}^{i,j}$, after merging them with HRM. We also collect 100 runs for a set of experiments in which those 15 HEMs and the anchor are observed in the same group. Thus, for each pair of HEMs (h^i and h^j), as well as h^a, we obtain their actual correlation $\rho^{i,j}$ from those measurements. This allows us comparing their real correlation $\rho^{i,j}$ with the estimated correlation after merging with HRM $\hat{\rho}^{i,j}$. In particular we measure the absolute distance (difference) between $|\rho^{i,j} - \hat{\rho}^{i,j}|$, so that the maximum difference obtained for a pair of HEMs is 2. This happens when the estimated correlation is 1 (or -1) and the real one is -1 (or 1).

C. Independence and identical distribution

HRM builds on these statistical properties for h^a, `PROCESSOR_CYCLES`, to apply order statistics. In practice, this holds since all values of h^a have been collected from the repeated execution of the same workload, with the same inputs, and enforcing the same hardware and software state as much as it can be controlled. We have further evaluated these properties quantitatively. We performed an ANOVA test [35] to assess identical distribution of `PROCESSOR_CYCLES` across sub-experiments. The result of the test is a p-value $p = 0.57$, so the test is not rejected comparing the law on the expected value of `PROCESSOR_CYCLES`, and tells us that the noise is identically distributed across sub-experiments. We assess independence within each sub-experiment with a Ljung-Box test [36] with lag $= 10$. With a significance level $\alpha = 0.05$, independence is not rejected in 96% of the sub-experiments, so measurements can be regarded as independent since the expectation is that the test is not rejected by a fraction of the tests matching $1 - \alpha$. Hence, we can use the order statistics for `PROCESSOR_CYCLES` after the merge as part of HRM because the noise is the same across sub-experiments and there is no dependence across values read.

D. Correlation between HEMs

In order to have reliable correlation estimates, we use the percentile bootstrap method [37] with the following methodology. We first compute bootstrap samples of size $n = 50$ for all sub-experiments; we compute the correlation between all pairs of HEMs; we then repeat $p = 100$ times those two steps and store the estimates of the correlation; for each pair of HEMs we have 100 estimates and we take the mean of those 100 values, which will be our reference estimation. We can also obtain the confidence interval for those 100 values for completeness.

For reference, we consider two other merging methods, referred to as *unsorted* and *sorted* respectively. The unsorted method simply concatenates results of different sub-experiments in the very same order they are collected, without analyzing any type of relationship between HEMs. The sorted method, instead, sorts the values collected for each HEM from lowest to highest merging in the same vector those in the same relative position for each HEM, so the lowest value for each HEM form a vector, the second lowest for each HEM another vector, and so on and so forth.

We have chosen workloads 1, 2, 5 and 10 due to their high variability and different spectra of measured values and correlations between HEMs in order to provide a representative set of cases. Figure 7 shows the correlation distance for the chosen workloads. Each point represents the absolute difference

[2]For the sake of convenience, we refer to the HEM read in the same sub-experiment as being in the same (HEM) group.

[3]All possible pairs with the 16 HEMs analyzed (the 15 relevant and h^a).

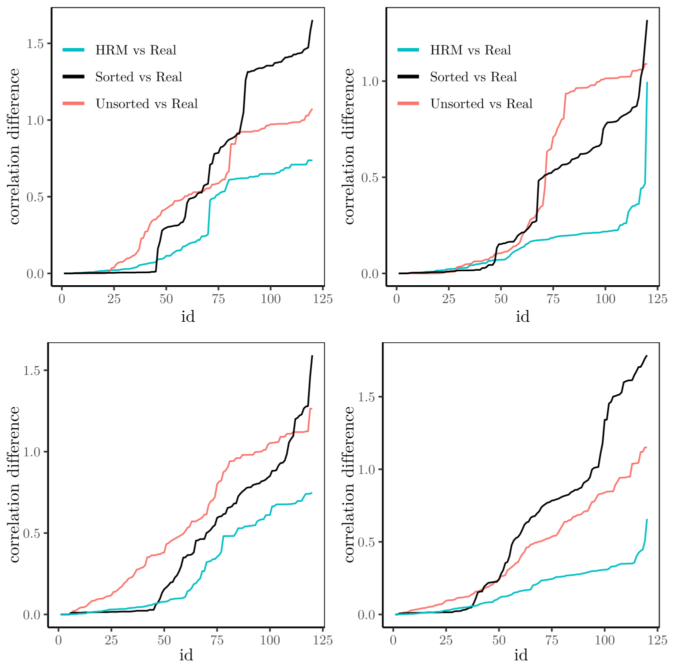

Fig. 7: Correlation Difference for each HEMs pair for workload 1 (top-left), workload 2 (top-right), workload 5 (bottom-left), and workload 10 (bottom-right).

between the estimated correlation for each method and the real correlation obtained measuring those HEMs in the same group. We order all HEM pairs for each method from lowest to highest correlation difference. As shown, the differences between the observed and estimated correlation for HRM is consistently lower than for the other methods, thus reflecting its higher accuracy. While the input data for all methods does not include direct observations of the real correlation and hence, such information is missing in statistical terms, HRM successfully recovers part of this information through their individual correlations with h^a, which is effectively observed.

The (naive) unsorted method is obviously poor and achieves good correlation only in some cases by chance.

The sorted method performs very well for those pairs of HEMs where both HEMs have strong positive correlation, since sorting them precisely joints correlated values. However, in many cases such correlation is either indirect or weak, which makes the sorted method particularly inaccurate leading to the highest discrepancies w.r.t. real correlations.

In the case of HRM, correlation is precisely estimated for those HEMs with significant correlation with h^a, since their mutual correlation is preserved with a probability matching the product of their individual correlations with h^a. However, if their individual correlation with h^a is weak at least for one of the HEM, their mutual correlation will be mostly lost, and the estimated correlation will approach 0. However, despite that, a key advantage of HRM is that joint correlation across HEMs is lost if and only if at least one of them is not meaningfully correlated with h^a. Instead, those correlations that matter for timing in our case, are preserved, as opposed to the other methods (unsorted and sorted), which preserve correlation for arbitrary pairs of HEM, not for those necessarily correlated with timing (i.e. h^a).

For the rest of the 12 workloads we cannot show such detailed results as for workloads 1 and 2. Instead we present a summarized analysis of the three methods for all workloads is shown in Table IV, in the form of the mean squared error (MSE). The MSE is the average of the squared errors, it is specifically computed as $\frac{1}{N}\sum_{i,j=1,\dots,nrh}^{i>j}(\rho^{i,j}-\hat{\rho}^{i,j})^2$, where nrh is the number of relevant HEMs, and N is the number of pairs $\binom{nrh}{2}$. As it can be seen, HRM shows to be the most accurate method sustainedly, and its accuracy is only relatively lower for Workload 4 since correlations with h^a in this workload are relatively weak in general.

TABLE IV: Mean squared error of the merging methods.

wld	HRM	Sorted	Unsorted	wld	HRM	Sorted	Unsorted
1	0.18	0.67	0.39	**9**	0.10	0.45	0.14
2	0.04	0.24	0.38	**10**	0.05	0.69	0.25
3	0.17	0.51	0.44	**11**	0.22	0.28	0.33
4	0.4	0.48	0.46	**12**	0.14	0.43	0.33
5	0.15	0.37	0.45	**13**	0.20	0.34	0.40
6	0.36	0.50	0.38	**14**	0.25	0.35	0.48
7	0.12	0.34	0.12	**15**	0.09	0.55	0.17
8	0.17	0.19	0.30	**16**	0.22	0.21	0.32

Correlation with the anchor. As stated, HRM aims at preserving the relationship between each HEM and the anchor. While such correlation is highly preserved by observing each HEM with h^a, HRM reduces the number of observations of h^a in each merged vector (nb, one for each run of each sub-experiment merged) by applying order statistics. Thus, only 1 HEM out of the $np-1$ in each merged vector preserves the actual value observed for h^a in its run, whereas the other $np-2$ have a different h^a value, which may have an effect on the correlation between HEMs and h^a. However, this effect is expected to be tiny. We assess this quantitatively in Figure 8 for workloads 1 and 2, where we show the estimated correlation (blue lines), the 95% confidence interval (red lines) and the real correlation (black dots). As expected, correlation is estimated with very high precision. We have observed this very same effect for all workloads and all pairs of HEMs, so we omit those data due to lack of further insights and of space.

E. Overheads

The HRM algorithm has very low computation requirements. To process the data of the experiments we performed in the T2080, the `R` implementation of HRM required less than 38 milliseconds on a Dell latitude e7490 laptop.

HRM requires nr runs for each of the nb sub-experiments, so a total of $nb \cdot nr$ runs. For instance, to read all 262 HEM, HRM required 53 sub-experiments to collect 5 different HEMs and h^a in each sub-experiment. Each sub-experiment was executed $nr = 100$ runs, thus above the minimum number of 30. Values for each workload were obtained in around 10 minutes. Note that, given that real-time programs usually last in the order of milliseconds, so few thousands of runs may only take up to few minutes in general.

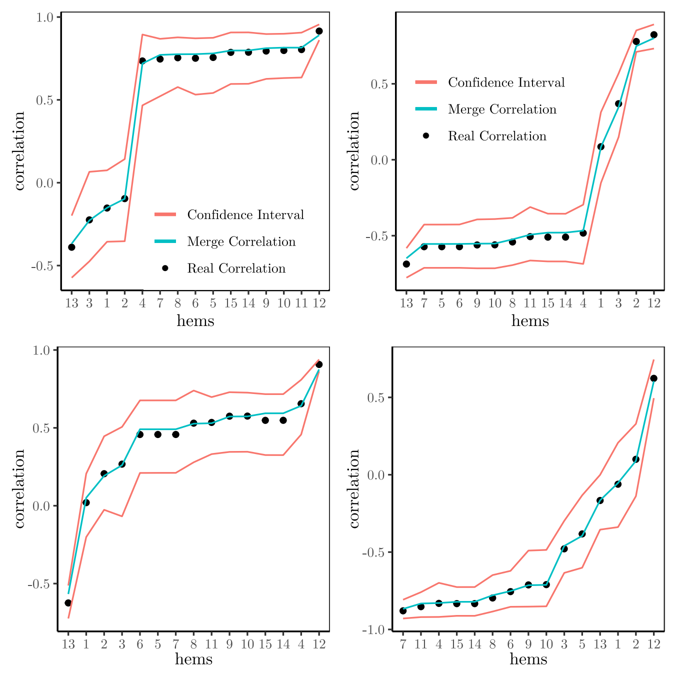

Fig. 8: Correlation between relevant HEMs and `PROCESSOR_CYCLES` before and after merging for workload 1 (top-left), workload 2 (top-right), workload 5 (bottom-left), and workload 10 (bottom-right).

VI. Related Works

In the real-time domain, several works build on HEMs for the estimation of bounds to software timing. Paulisch et al. [21] create an analysis and runtime monitoring solution for limiting task contention in multicores by tracking and controlling HEM. In the same vein, Diaz et al. [38] build on HEM to produce an ILP-based contention model for an AURIX automotive microcontroller. Likewise, Santinelli et al.[39], build on the HEM of a multicore system to derive probabilistic WCET estimates. Griffin et al. [40] derive a method to select the HEM with highest contribution to software timing and predict execution time under unseen configurations.

More recently, information from HEMs has been exploited as the cornerstone of industrial-quality approaches [16], [41] for providing the necessary evidence for supporting the certification of multicore CES, in conformance with the requirements from domain-specific certification authorities [42].

Several works in the mainstream (high-performance) domain reason on the sources of variability in HEM values when executing several times the same piece of software. This covers from the operating system noise [43], application variability [44], [45] and the particular HEM-reading library, to the complexity of the hardware [46]. For instance, [43] focuses on the cycle count HEM and shows that its variability is often related to the executable layout and operating system issues. Also, at software level, [46] assesses the accuracy of various high-level counter APIs with focus on cycle count and total retired instruction HEMs. In our work, we use no operating system and access directly, with no library, the HEMs (via the PMCs) so they are not subject to software-induced variability. In [45], authors focus on task-parallel programs in high-performance environments with highly-dynamic execution conditions, including dynamic task scheduling, that cause tasks to execute in different orders and in different cores across executions. Authors propose techniques to determine which HEM readings belong to each task and hence, combine them to derive all HEMs for a task. Interestingly, the reading of each group of *nP* HEMs is performed once, so authors do not assess the impact of variability in HEM readings due to hardware and software related variability. We, instead, focus in much more predictable environments, as needed for CES and consider the variability of HEM readings.

HEM sampling or multiplexing consists in time-sharing the PMCs over a set of HEMs: at each interval boundary, whose duration is a configuration parameter, the PMCs are reprogrammed to read a different set of HEMs. HEM sampling is, for instance, adopted by the Linux kernel's perf event subsystem. The potential inaccuracies introduced by the interpolation made by sampling techniques have been studied elsewhere [47], [48]. In this paper, we do not perform any multiplexing of HEMs as it causes having phases of the program in which particular HEMs are not read, resorting to interpolation methods. Instead, HRM builds upon HEM values from the observation of the full program execution, without any kind of interpolation.

At hardware level, other works [27] focus on specific HEMs (e.g. retired instructions, branches, loads/stores) and develop low-level hardware hypotheses on the reasons behind some of these HEMs suffering from various forms of under and over count. Authors do several recommendations on the hardware design to reduce the observed variability. Our goal, instead, is managing at software level HEM variability and limitations to read HEMs on existing boards (e.g. NXP T2080).

Incomplete sets of data have been considered with Matrix Completion (MC) methods [32], [33]. There are fundamental differences between HRM and MC. HRM aims at merging *only* observed data, with no assumption on the distribution of the input data to merge. Conversely, (1) MC requires input data be a random matrix, where values in all rows and columns belong to one measure with its own distribution for each value, which does not hold for the problem at hand (e.g. the outcome of a HEM measure is a column, thus with its own distribution). (2) MC aims at producing synthetic data to complete missing data, thus bringing risks due to inferring the distribution of real data to produce new data, which may match some characteristics of real input data but miss others. In fact, since input data (HEM values) do not match requirement (1), and the fraction of missing values is large (np observed values out of nh HEMs, where $np << nh$), MC populates the data array with values whose mean and standard deviation differs drastically from those for real (observed) data. Overall, MC does not fit the needs of the problem at hand by construction since its prerequisites are not met.

VII. Conclusions

Measurement-based timing analysis methods increasingly build on HEMs to measure and estimate the timing behavior of time-critical applications running on MPSoCs. Unfortunately, in complex MPSoCs HEM values are subject to some unavoidable noise, and they can only be read in small subsets, thus allowing end users only to collect partial snapshots (i.e. including only a subset of HEMs) subject to different and unknown noise. Therefore, end users address the challenge of combining all HEM values, as a naive merging could lead to inconsistent joint values. This paper presents HRM, a flexible method to merge HEM values across runs that allows preserving precisely their correlation with timing and preserving, to a good extent, their joint correlation. HRM achieves its goals by building on (1) non-parametric statistics, which do not pose any constraint on the distributions observed for different HEMs in the T2080, and (2) the use of an anchor HEM to relate measurements from different HEMs. Our evaluation on a complex MPSoC – the NXP T2080 – targeting commercial avionics systems validates HRM, showing that it outperforms other approaches to merge HEM values.

Acknowledgements

This work has been partially supported by the Spanish Ministry of Science and Innovation under grant PID2019-107255GB, the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 772773) and the HiPEAC Network of Excellence.

References

[1] Xilinx, 2020. [Online]. Available: https://www.nvidia.com/en-us/self-driving-cars/drive-platform/hardware/

[2] ——, "Rockwell Collins Uses Zynq UltraScale+ RFSoC Devices in Revolutionizing How Arrays are Produced and Fielded: Powered by Xilinx," 2019. [Online]. Available: https://www.xilinx.com/video/corporate/rockwell-collins-rfsoc-revolutionizing-how-arrays-are-produced.html

[3] S. Grigorescu et al., "A survey of deep learning techniques for autonomous driving," *ArXiv*, vol. abs/1910.07738, 2019.

[4] R. Wilhelm and J. Reineke, "Embedded systems: Many cores — many problems," in *SIES*, 2012.

[5] J. Reineke, "Challenges for timing analysis of multi-core architectures," Workshop on Foundational and Practical Aspects of Resource Analysis, 2017, invited Talk.

[6] R. Mancuso et al., "Real-time cache management framework for multi-core architectures," in *RTAS*, 2013.

[7] T. Kloda et al., "Deterministic memory hierarchy and virtualization for modern multi-core embedded systems," in *RTAS*. IEEE, 2019.

[8] L. Liu et al, "A software memory partition approach for eliminating bank-level interference in multicore systems," in *PACT*, 2012.

[9] H. Yun et al, "PALLOC: DRAM bank-aware memory allocator for performance isolation on multicore platforms," in *RTAS*, 2014.

[10] X. Pan and F. Mueller, "Controller-aware memory coloring for multicore real-time systems," in *SAC*. ACM, 2018, pp. 584–592.

[11] N. Suzuki et al., "Coordinated bank and cache coloring for temporal protection of memory accesses," in *CSE*, 2013.

[12] R. Pellizzoni et al., "A predictable execution model for cots-based embedded systems," in *RTAS*, 2011.

[13] A. Biondi and M. D. Natale, "Achieving predictable multicore execution of automotive applications using the LET paradigm," in *RTAS*, 2018.

[14] DDC-I, "Patent Details for Managing Cache," https://www.ddci.com/manage_cache_patent/, 2020.

[15] P. K. Valsan et al, "Taming non-blocking caches to improve isolation in multicore real-time systems," in *RTAS*, 2016, pp. 161–172.

[16] D. Radack et al. (Rockwell Collins), "Civil Certification of Multi-core Processing Systems in Commercial Avionics," 2018.

[17] B. Dreyer et al., "Continuous non-intrusive hybrid WCET estimation using waypoint graphs," in *WCET workshop*, 2016.

[18] A. Betts et al., "Hybrid measurement-based WCET analysis at the source level using object-level traces," in *WCET workshop*, 2010.

[19] K. Schmidt et al., "Non-intrusive tracing at first instruction," in *SAE Technical Paper*. SAE International, 04 2015.

[20] J. Jalle et al., "AHRB: A high-performance time-composable AMBA AHB bus," in *RTAS*, 2014.

[21] J. Nowotsch et al., "Multi-core interference-sensitive wcet analysis leveraging runtime resource capacity enforcement," in *ECRTS*, 2014.

[22] H. Yun et al., "Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms," in *RTAS*, 2013.

[23] R. Pellizzoni et al., "Worst case delay analysis for memory interference in multicore systems," in *DATE*, 2010.

[24] D. Dasari et al, "Response time analysis of cots-based multicores considering the contention on the shared memory bus," in *TrustCom*, 2011.

[25] NXP, "QorIQ® T2080 and T2081 Multicore Communications Processors."

[26] Freescale semicondutor, "QorIQ T2080 Reference Manual," 2016, Also supports T2081. Document Number: T2080RM. Rev. 3, 11/2016.

[27] V. Weaver et al., "Non-determinism and overcount on modern hardware performance counter implementations," in *ISPASS*, 2013.

[28] Freescale semicondutor, "Arm® Architecture Reference Manual. Armv8, for Armv8-A architecture profile," ARM DDI 0487F.b (ID040120).

[29] Infineon, *AURIX™ TC29x B-Step 32-Bit Single-Chip Microcontroller - User's Manual V1.3 2014-12*, 2019.

[30] R. Hyndman and Y. Fan, "Sample quantiles in statistical packages," *The American Statistician*, vol. 50, no. 4, pp. 361–365, 1996.

[31] D. Freedman et al., *Statistics: Fourth International Student Edition*, ser. International student edition. W.W. Norton & Company, 2007.

[32] B. Recht, "A simpler approach to matrix completion," *J. Mach. Learn. Res.*, vol. 12, no. null, p. 3413–3430, Dec. 2011.

[33] T. Hastie et al., "Matrix completion and low-rank svd via fast alternating least squares," *J. Mach. Learn. Res.*, vol. 16, no. 1, Jan. 2015.

[34] Freescale semicondutor, "e6500 Core Reference Manual," https://www.nxp.com/docs/en/reference-manual/E6500RM.pdf, 2014, E6500RM. Rev 0. 06/2014.

[35] R. A. Fisher, "Xv.—the correlation between relatives on the supposition of mendelian inheritance." *Transactions of the Royal Society of Edinburgh*, vol. 52, no. 2, p. 399–433, 1919.

[36] G. M. Ljung and G. E. P. Box, "On a measure of lack of fit in time series models," *Biometrika*, vol. 65, no. 2, pp. 297–303, 08 1978.

[37] A. C. Davison and D. V. Hinkley, *Bootstrap Methods and their Application*, ser. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1997.

[38] E. Díaz et al., "Modelling multicore contention on the aurix tc27x," in *DAC*, 2018.

[39] F. Guet et al., "Probabilistic analysis of cache memories and cache memories impacts on multi-core embedded systems," in *SIES*, 2016.

[40] D. Griffin et al., "Forecast-based interference: Modelling multicore interference from observable factors," in *RTNS*, 2017.

[41] S. H. VanderLeest and C. Evripidou, "An approach to verification of interference concerns for multicore systems (CAST-32A)," in *SAE Technical Paper*. SAE International, 2020.

[42] Federal Aviation Administration, Certification Authorities Software Team (CAST), *CAST-32A Multi-core Processors*, 2016.

[43] T. Mytkowicz et al., "Producing wrong data without doing anything obviously wrong!" in *ASPLOS*, 2009.

[44] A. R. Alameldeen and D. A. Wood, "Variability in architectural simulations of multi-threaded workloads," in *HPCA*.

[45] R. Neill et al., "Fuse: Accurate multiplexing of hardware performance counters across executions," *TACO*, vol. 14, no. 4, 2017.

[46] D. Zaparanuks et al., "Accuracy of performance counter measurements," in *ISPASS*, 2009.

[47] R. V. Lim, "Computationally efficient multiplexing of events on hardware counters," in *Linux Symposium*, 2014.

[48] H. Xu et al., "Can we trust profiling results?: understanding and fixing the inaccuracy in modern profilers," in *Proceedings of the ACM International Conference on Supercomputing, ICS 2019*, 2019.