MUCH: Exploiting Pairwise Hardware Event Monitor Correlations for Improved Timing Analysis of Complex MPSoCs

Sergi Vilardell[†*], Isabel Serra[†§], Enrico Mezzetti[†], Jaume Abella[†], and Francisco J. Cazorla[†]

Barcelona Supercomputing Center, Spain[†]
Universitat Politècnica de Catalunya, Spain[*]
Centre de Recerca Matemàtica, Spain[§]

ABSTRACT

Measurement-based timing analysis techniques increasingly rely on the Performance Monitoring Units (PMU) of MPSoCs, as these units implement specialized Hardware Event Monitors (HEMs) that convey detailed information about multicore interference in hardware shared resources. Unfortunately, there is an evident mismatch between the large number of HEMs (typically several hundreds) and the comparatively small number (normally less than ten) of Performance Monitoring Counters (PMCs) that can be configured to track HEMs in the PMU. Timing analysis normally require to observe a non-negligible number of HEMs per task from the same execution. However, due to the small number of PMCs, HEMs are necessarily collected across multiple runs that, despite intended to repeat the same experiment, carry out some significant variability (above 50% for some HEMs in relevant MPSoCs) caused by platform-intrinsic execution conditions. Therefore, blindly merging HEMs from different runs is not acceptable since they may easily correspond to significantly different conditions. To tackle this issue, the HRM approach has been proposed recently to merge HEMs from different runs accurately preserving their correlation w.r.t. one anchor HEM (i.e. processor cycles) building on order statistics. However, HRM do not always preserves the correlation between other pairs of HEMs that might be lost to a large extent. This paper copes with HRM limitations by proposing the MUlti-Correlation HEM reading and merging approach (MUCH). MUCH builds on multivariate Gaussian distributions to merge HEMs from different runs while preserving pairwise correlations across each individual pair of HEMs simultaneously. Our results on an NXP T2080 MPSoC used for avionics systems show that MUCH largely outperforms HRM for an identical number of input runs.

ACM Reference Format:
Sergi Vilardell, Isabel Serra, Enrico Mezzetti, Jaume Abella, and Francisco J. Cazorla. 2021. MUCH: Exploiting Pairwise Hardware Event Monitor Correlations for Improved Timing Analysis of Complex MPSoCs. In *The 36th ACM/SIGAPP Symposium on Applied Computing (SAC '21), March 22–26, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3412841.3441931

SAC '21, March 22–26, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8104-8/21/03…$15.00
https://doi.org/10.1145/3412841.3441931

1 INTRODUCTION

The pervasive adoption of increasingly autonomous systems in domains such as automotive and avionics impose the use of multiprocessor system-on-chips (MPSoCs) to reach the performance levels required. However, while the use of MPSoCs increases in those domains [32], they also hinder software timing analysis as timing bounds become inherently dependent on multicore interference in the access of shared hardware resources like shared caches, memory controllers, buses, and on-chip interconnects [10, 19, 35, 41].

Several solutions mitigate multicore interference by exploiting time segregation [3, 6, 31] (e.g. by partitioning applications into memory and computing phases) or space segregation (e.g. making different tasks access different hardware blocks) [8, 11, 16, 20–22, 29, 37]. The former approach is potentially intrusive with the application code and semantics, and thus is not applicable in many cases. The latter does not avoid multicore interference altogether that can still occur in hardware resources not visible at software level, such as interconnects, as well as buffers and internal queues to shared caches [4, 28].

Powerful measurement-based timing analysis solutions have been proposed for multicores for critical applications building on hardware and software profiling [2, 9, 36]. In this context, the Performance Monitoring Unit (PMU) in MPSoCs offers information relevant for timing analysis [23], enabling verification and validation (V&V) and software time budgeting of time-critical applications. For instance, the usage of some shared resources can be budgeted, monitored, and enforced using event quotas building on Hardware Event Monitors (HEMs) reached through the PMU [5, 27, 30, 43]. HEMs are used to monitor when tasks exceed their usage quotas which are suspended. Indeed, HEM information is already used as a pillar to certify critical avionics systems [33], so PMUs, and the HEMs they allow monitoring, become the basis of industrial-quality multicore interference mitigation and estimation techniques.

PMUs typically support hundreds of HEMs, often related to the access counts for different types of accesses and to different hardware shared resources. However, HEMs can only be interfaced through a much lower number (typically less than 10) of user-visible performance monitoring counters (PMCs). Therefore, when the number of HEMs relevant for timing analysis is higher than the number of PMCs, HEMs need to be read in multiple experiments.

For instance, the NXP T2080 MPSoC has around 20 HEMs just for monitoring the shared L2 cache and only 6 PMCs.

This limitation confronts with the fact that task scheduling needs to budget how much each task is expected to access shared resources, which can only be done, in measurement-based approaches, by reading a large number of HEMs simultaneously or, at least, consistently. However, collecting HEMs by performing several runs has some notable side effects on how the gathered values can be consistently merged: even if the very same experiment is intended to be repeated, the impossibility of fully controlling the hardware and software initial state brings significant variations across runs. A variation up to 59% in processor cycles has been observed on the NXP T2080 MPSoC [39], despite the same program is executed program with the same inputs. Hence, gathering a comprehensive set of information by merging different HEMs from different runs in a consistent way turns out to be a challenging task.

Recently, HRM [39] has been proposed to tackle this challenge. HRM builds on order statistics to relate HEMs from different runs through an anchor HEM (typically processor cycles) to merge HEMs while preserving their mutual correlations; so that the final merged HEM vectors resemble as much as possible those that would have been obtained when the number of PMCs was high enough to read all HEMs at once. While HRM efficiently preserves the correlation of each HEM with the anchor HEM, it only exercises indirect control on the correlation between other HEMs leading to limited accuracy. Moreover, accuracy cannot be improved by collecting further HEM measurements due to the intrinsic characteristics of HRM, thus challenging those methods that require consistent reads of a large number of HEM values.

Contribution: We propose MUCH, a MUlti-Correla-tion HEM reading and merging approach. MUCH overcomes the limitations of HRM with a completely different strategy to collect HEM values and using multivariate Gaussian distributions to merge HEM values, drastically improving the accuracy of the merged HEM vectors for all pairs of HEMs at once. In particular our contributions are:

(1) We identify the intrinsic limitations of HRM to preserve correlations across HEMs if the correlation of any of them with the anchor HEM is weak, which may lead to HEM vectors with limited accuracy.
(2) As part of MUCH, we propose a different strategy to collect HEM values that, instead of observing all HEMs together with an anchor, ensures that each pair of HEMs is observed together. This guarantees capturing useful information to preserve pairwise HEM correlations, and ultimately allows preserving all pairwise HEM correlations.
(3) The core of MUCH is the arrangement and merging of individual and independent HEM readings into a single dataset as if they were measured together with the use of multivariate Gaussian distributions (MVGD) which allow us to preserve all pairwise HEM correlations simultaneously.
(4) We evaluate MUCH and compare it with HRM. Our results on an NXP T2080 MPSoC considered for avionics systems show that (i) MUCH outperforms HRM by preserving all pairwise HEM correlations regardless of their individual correlations with any HEM, which is a limitation of HRM; and (ii) as we increase the number of measurements collected, MUCH increases its accuracy reaching nearly perfect correlations, as opposed to HRM, which quickly plateaus due to the lack of information and ability to preserve some pairwise HEM correlations.

The rest of this paper is organized as follows. Section 2 introduces the need for merging HEM values into complete HEM vectors. Section 3 formalizes the problem tackled. Section 4 presents MUCH, our solution for HEM merging. MUCH evaluation and comparison against HRM is provided in Section 5. Section 6 reviews the related work. Section 7 draws the main conclusions of this work.

2 MOTIVATION

Magnitude of the variability. HEM variability in the NXP T2080 MPSoC [13] has been shown to be high [39]: when running a 4-task workload with each task pinned to a specific core and collecting HEMs all 262 HEMs in 44 different sub-experiments (there are 6 PMCs in the T2080), the observed HEM variability is high. Despite ignoring HEMs whose order of magnitude is low to have a relevant impact in the execution time (processor cycles), the variability observed for some HEMs is above 60%, with variability on processor cycles up to 45% variability (reaching 59% variability in other experiments for other workloads). Interestingly, HEMs related to the user-visible functional behavior of the program (e.g. total and per-type instruction counts) remain constant, reflecting that the very same task is run with exactly the same inputs on each run. Hence, the source for such (high) variability is the hardware platform itself [39].

Distribution of the variability. The observed variability is not just related to few outliers, but concerns the big bulk of the distributions [39], which precludes the use of simple approaches to discard few outliers and retain all the other measurements. Distributions observed for the HEMs with a relevant order of magnitude include Normal, concave, convex and clustered distributions, along with other distributions hard to fit into any specific category. It is of prominent importance the fact that the HEM `PROCESSOR_CYCLES`, the target of timing analysis methods, has a distribution hard to fit into any specific category.

Sources of the variability. MPSoCs such as the NXP T2080 and the Xilinx Zynq UltraScale+ offer high degree of parallelism. As a result, several parallel activities occur simultaneously such as, for instance, multiple instructions being executed in the different cores, DRAM refreshes, pending core requests in queues and buffers in shared caches, interconnects and memory controllers, etc. Analogously, MPSoCs carry out plenty of state information in a large number of components such as replacement history in all cache memories, initial state of buffers and queues in all on-chip components, branch predictors state, etc.; with most of the elements contributing to such state being not explicitly controllable by software means. Therefore, any attempt to re-run the very same software with the very same input data is subject to many sources of variability. While their impact is limited individually (e.g. making core 1 accessing a shared cache before core 2), repeated access patterns in software may cause such impact to repeat systematically, thus leading to large relative variability for many HEMs, including `PROCESSOR_CYCLES`. Moreover, it is generally impossible to exercise

Table 1: Main terms used in this work.

Term	Definition
nh	Number of HEMs.
np	Number of PMCs.
nb	Number of sub-experiments.
nr	Number of runs per sub-experiment.
N	Total number of runs.
$\mathcal{H}$	Set of HEMs.
h^i	HEM with id i.
$\{h^i\}$	Vector of concatenated samples of h^i in different subexperiments.
r_k	Run k as an observed vector of dim np.
ρ_{ij}	Correlation between HEMs h^i and h^j.
S	Correlation matrix.
σ_i	Variance of HEM i.
σ_{ij}	Covariance between HEMs i and j.
Σ	Covariance matrix.
Σ_0	Covariance matrix obtained from transformed data.
$\mathcal{N}_{nh}(\mu, \Sigma)$	nh-dimensional Gaussian distribution with vector of means μ and covariance matrix Σ.
$\hat{x}$	Empirical version of variable x.
x'	Optimized computation of variable x.
x_{hrm}	Variable pertaining HRM.
x_{much}	Variable pertaining MUCH.

full control on those sources of variability, or to be able to identify the particular combination of factors affecting each individual run.

Number of PMCs and HEMs. For cost reasons, MPSoCs implement only few PMCs, typically in the range of 2-8 per core as this reduces the routing logic (e.g. multiplexors and wires) needed to connect HEMs (in the order of hundreds) and PMCs. This trend is common in MPSoCs like the Xilinx UltraScale+ or the NXP T and L families. For some SoCs, e.g. the Infineon AURIX, it is common having HEMs that can only be mapped to a subset of the PMCs, further constraining the HEMs that can be read simultaneously.

Correlation among HEMs. Keeping the correlation among HEMs read in different runs is instrumental for timing analysis to provide clear and comprehensive insight on how running tasks use all shared resources, by considering the values in their respective HEMs together (e.g. caches, buses and memory). Correlation also helps deriving other metrics relevant for the analysis that combine different HEMs, like cache misses per thousand instructions or cycles, memory misses per cache access, and alike.

3 PROBLEM FORMALIZATION

Our goal is collecting values for a given set of HEMs, regarded as relevant for timing analysis, $\mathcal{H} \ni \{h^1, h^2, \cdots, h^{nh}\}$ for a given program running on MPSoC, with a given input data set. We use the terminology shown in Table 1.

In an MPSoC where variability occurs across measurements the ideal scenario corresponds to the case where the number of HEMs needed (nh) does not exceed the number of PMCs in the platform (np). In this case, we just need to collect as many measurements (runs) as needed to capture variability to a sufficient extent, as dictated by the timing analysis method used, and each run captures all HEMs at once (all $h^i \in \mathcal{H}$), as illustrated in the example in Figure 1. Note that the number of measurements required solely depends on the timing analysis method used, and it is out of the scope of this work, where we focus on how to provide the timing analysis with observations providing consistent data for all HEMs in each HEM vector.

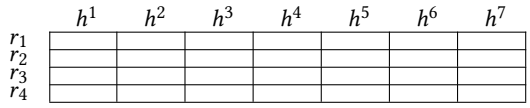

Figure 1: Runs collected with an ideal MPSoC with $np \geq nh$. In particular, $np = nh = 7$ in the example.

As discussed in Section 2, the number of PMCs is rather low, and will be typically lower than the number of HEMs relevant for the timing analysis. Formally stated, $np < nh$. In that case, each complete set of HEMs needs to be obtained with multiple runs (sub-experiments), and the actual HEMs to read on each run are determined by the method used to merge them. The lowest number of runs needed to collect all HEMs is given by $nb_{min} = \lceil \frac{nh}{np} \rceil$. In the case of HRM, for instance, the *anchor HEM* (i.e. processor cycles typically) is read on each run, because it is used as the link across runs reading different HEMs to merge them into a single HEM vector. Thus, in the case of HRM, the number of sub-experiments is $nb_{hrm} = \lceil \frac{nh-1}{np-1} \rceil$. Figure 2 illustrates the case for HRM when $np = 3$ and $nh = 7$, and h^1 is the anchor HEM. As shown, $nb = 3$, to read h^1, h^2 and h^3 in sub-experiment 1, h^1, h^4 and h^5 in sub-experiment 2, and h^1, h^6 and h^7 in sub-experiment 3. For the sake of illustration, we set the number of runs per sub-experiment $nr = 4$. Therefore, the total number of runs $N = nb \cdot nr = 12$.

Note that, in our example, we assume that all HEMs can be read with all PMCs, so that there is no constraint to allocate HEMs to PMCs in each sub-experiment. If some HEMs can only be read with a subset of PMCs, this could add further constraints and, potentially, lead to a higher number of sub-experiments. This type of constraints is platform-dependent and does not change the challenges and solutions presented in this paper.

4 MUCH: MULTI-CORRELATION HEM READING AND MERGING

On each run where a subset of HEMs is read (np out of nh), the particular platform state which determines the actual values read is – as discussed before – to some extent unknown, uncontrollable and, due to the many combinations of multiple possible platform factors, practically unique. This uniqueness makes that only HEMs read in the same sub-experiment can be guaranteed to be exposed to an identical platform state in each run, which we refer to as noise for concision. Therefore, HEM values read in different sub-experiments are necessarily exposed to different noise. Still, some relation, stronger or weaker, exists among many HEMs (i.e. their pairwise correlation), and by preserving such relation, HEMs can be merged in a way that they correspond to similar noise levels.

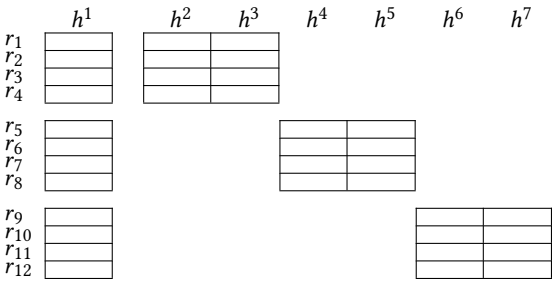

Figure 2: Runs collected with HRM when $np < nh$. In particular, $np = 3$ and $nh = 7$ in the example.

Formally stated, every pair of HEMs h^i and h^j exhibits some degree of correlation ρ_{ij} when they are read in the same sub-experiment. However, if read across different sub-experiments, we have different sets of values for each HEM, and the challenge is to merge them into complete HEM vectors so that the pairwise correlations among each h^i and h^j pair is as close as possible to ρ_{ij}.

4.1 HRM Analysis

The approach followed by HRM consists in reading an anchor HEM, h^a in each sub-experiment along with $np - 1$ other HEMs, without repeating any of the other $nh - 1$ HEMs across sub-experiments. See Figure 3 for a visual explanation of HRM with two sub-experiments and the first column representing the anchor. Then, it merges all HEMs into complete HEM vectors by preserving their correlation with the anchor h^a perfectly, since all HEMs are read along with h^a in one sub-experiment. That is, for any HEM h^i, the correlation ρ_{ai} is fully preserved. In fact, if two HEMs, h^i and h^j, where $i \neq a$ and $j \neq a$, are read in the same sub-experiment, they will be merged together in the merged HEM vector and hence, their pairwise correlation ρ_{ij} will also be fully preserved. However, no action is taken to preserve the correlation among any other pair of HEMs read in different sub-experiments. Recalling the example in Figure 2, and focusing on a specific HEM (e.g. h^2) this implies that correlations ρ_{21} and ρ_{23} are fully preserved, whereas correlations ρ_{24}, ρ_{25}, ρ_{26}, and ρ_{27} are not. Since those HEMs in different sub-experiments are placed in the same merged HEM vector through their relation with h^a, as discussed before, their pairwise correlation is only preserved to a limited extent dictated by ρ_{ai} and ρ_{aj}. For instance, in our example, correlation ρ_{24} is preserved as much as determined by the product $\rho_{21} \cdot \rho_{41}$. If such product is low, then HRM will preserve ρ_{24} poorly, and increasing the number of runs for each sub-experiment cannot cure such limitation.

Formalization. When two strongly correlated HEMs, h^x and h^y, with correlation ρ_{xy}, are read in different sub-experiments, then, given an anchor HEM h^a, HRM preserves ρ_{xy} correlation only partially $\rho_{xy}^{factor} = \rho_{ax} \cdot \rho_{ay}$. Hence, under HRM the actual correlation preserved is $\rho_{xy}^{hrm} = \rho_{xy}^{factor} \cdot \rho_{xy}$. Unless both ρ_{ax} and ρ_{ay} are very high (e.g. close to 1.0), the degree of joint correlation preserved drops significantly and their actual correlation in the merged HEM vector can be drastically different to the real one. For instance, if $\rho_{xy} \approx 1.0$, but $\rho_{ax} \approx 0.7$ and $\rho_{ay} \approx 0.7$, then $\rho_{xy}^{factor} < 0.5$ and hence, $\rho_{xy}^{hrm} < 0.5$, largely below $\rho_{xy} \approx 1.0$.

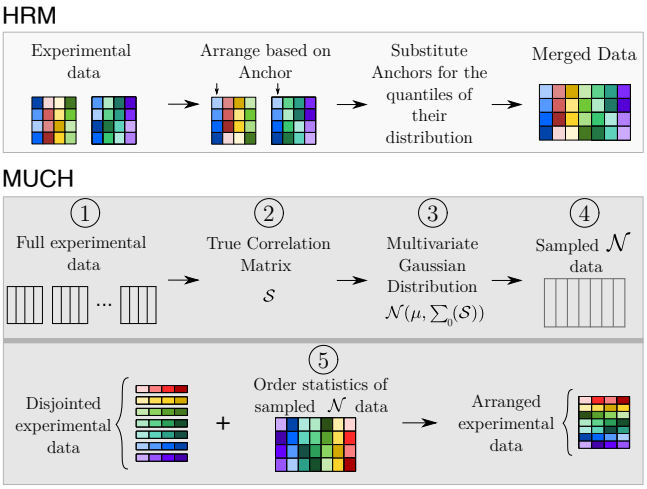

Figure 3: Diagrams of HRM and MUCH methodologies. On both diagrams the colors represent the order statistics of each column, i.e. lighter colors represent lower values while darker colors represent higher values.

Such limitation is intrinsic to the way HRM collects sub-experiments, since, in general the correlation across HEMs in different sub-experiments is not observed (as HRM resorts only the relations of each HEM with the anchor HEM). As a result, even if HRM collected more runs per sub-experiment or even different sub-experiments, it would still be unable to get rid of its intrinsic limitation.

Our proposal, MUCH, tackles explicitly this issue by observing all HEM pairwise correlations through a set of appropriately designed sub-experiments, and by using multivariate Gaussian distributions to preserve those correlations when merging HEM values into complete HEM vectors. In the next section, we deepen into HRM limitations and present MUCH.

Summary. HRM preserves the correlation between each HEM and the anchor is preserved with almost perfect accuracy in the merged HEM vector. The number of sub-experiments needed by HRM, nb_{hrm} is typically very close to the minimum possible number of sub-experiments nb_{min}. However, in general, HRM does not collect any data to retrieve the degree of correlation between HEMs in different sub-experiments. Therefore, the merged HEM vector may completely lose the correlation among strongly correlated HEMs read in different sub-experiments.

4.2 Introduction to MUCH

MUCH, instead, builds upon observing all pairwise correlations across HEMs. Then, MUCH merges values from the different HEMs into complete HEM vectors preserving all pairwise correlations *simultaneously*. For instance, given 4 HEMs h^1, h^2, h^3 and h^4, MUCH organizes their values into complete HEM vectors so that preserving some correlations (e.g. all correlations with h^1, $\langle h^1, h^2 \rangle$, $\langle h^1, h^3 \rangle$ and $\langle h^1, h^4 \rangle$) does not impact negatively the other correlations (e.g. $\langle h^2, h^3 \rangle$, $\langle h^2, h^4 \rangle$ and $\langle h^3, h^4 \rangle$). Therefore, MUCH's challenge is putting all HEM values into complete vectors so that all pairwise correlations observed across HEMs *separately* are preserved *simultaneously* after merging.

Interestingly, by observing all pairwise HEM correlations one does not overcome the problem of preserving those correlations. It is still necessary to devise a proper method to merge all the observed readings into a complete dataset while keeping the correlations as close as possible to the ideal scenario. For instance, in the example above, if we merge all HEMs preserving their perfect correlation with h^1, e.g. sorting by h^1 values in all pairs $\langle h^1, h^2\rangle$, $\langle h^1, h^3\rangle$ and $\langle h^1, h^4\rangle$, and merging directly the rows keeping any of the values of h^1 in each row, we would keep perfect correlation between h^1 and each other HEM. However, we would likely degrade significantly the correlation of the pairs $\langle h^2, h^3\rangle$, $\langle h^2, h^4\rangle$ and $\langle h^3, h^4\rangle$. The challenge addressed by MUCH is exactly this one: *how much do I have to "sacrifice" each pairwise correlation so that, when putting all HEMs together, each pairwise correlation is still close to the observed one?*

To observe all pairwise correlations across HEMs, sub-experiments need to be arranged so that all pairs of HEMs are observed together in at least one sub-experiment. This is illustrated in Figure 4 for the same example discussed for HRM, with $np = 3$ and $nh = 7$. As shown, for instance, h^2 is observed in the first sub-experiment (r_1-r_4) together with h^1 and h^3, in the forth sub-experiment (r_{13}-r_{16}) together with h^4 and h^6, and in the fifth sub-experiment (r_{17}-r_{20}) together with h^5 and h^7. This allows *observing* all pairwise correlations, and thus, organizing observed values into merged HEM vectors where $\hat{\rho}_{ij} \sim \rho_{ij}$ for each pair of HEMs h^i and h^j.

Given that we have observed all pairwise HEM correlations, MUCH generates a nh-dimensional model of the HEMs by building on MVGD. Then, by organizing HEM values read according to that model, MUCH generates merged HEM vectors where all pairwise HEM correlations are preserved simultaneously with high accuracy, as detailed in next subsection.

Also MUCH requires a higher number of sub-experiments than HRM to observe all pairwise HEM correlations. Such number increases with the number of relevant HEMs (nh) and whenever the number of PMCs (np) decreases. However, as shown later in the evaluation section, under similar number of runs for HRM and MUCH, i.e. $nb_{hrm} \cdot nr_{hrm} \approx nb_{much} \cdot nr_{much}$, MUCH provides higher accuracy. Note that in that case, typically we have $nb_{hrm} < nb_{much}$ and $nr_{hrm} > nr_{much}$.

4.3 Mathematical approach

By virtue of the Central Limit Theorem, we can model the expected value of each HEM as a Gaussian distribution. This can be extended to multiple variables (e.g. the nh HEMs of interest), which are regarded as a MVGD with nh dimensions modeling the expected value of each HEM.

Definition. Let us consider a set of l i.i.d. Gaussian random variables $Z \sim N(0, 1)$. The covariance matrix for Z is the identity matrix I_l with expected value 0. Now, let A be a $k \times l$ matrix and μ be a k-vector, both with real finite coefficients. Then, $X = AZ + \mu$ has a MVGD. The expected value of X is μ and the covariance matrix is $\Sigma = AA^T$. The usual way of writing the MVGD is $X \sim \mathcal{N}(\mu, \Sigma)$ [42].

The computation of this model requires the expected value and the variance of each HEM, μ and σ^2, and the covariance matrix Σ, where the latter can be obtained from the correlation matrix S. In particular, the covariance of variables X and Y, σ_{XY}, can be obtained as follows, where ρ_{XY} is their correlation.

$$\sigma_{XY} = \rho_{XY} \cdot \sigma_X \cdot \sigma_Y$$

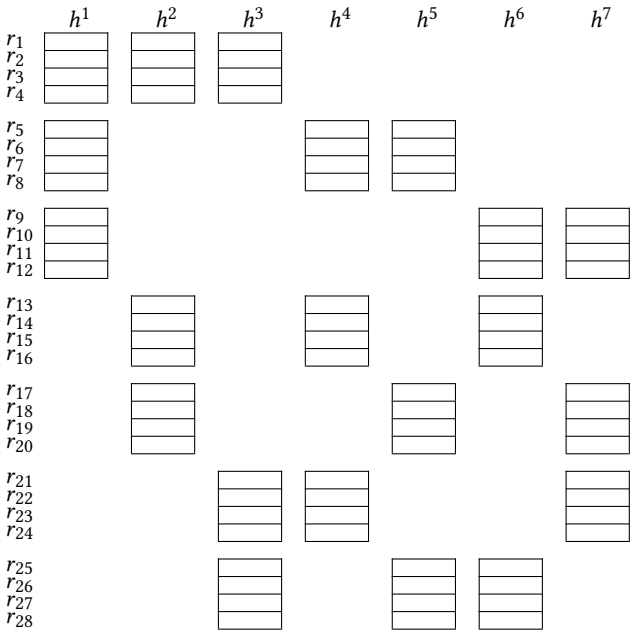

Figure 4: Runs collected with MUCH when $np < nh$. In particular, $np = 3$ and $nh = 7$ in the example.

The measurement collection procedure allows obtaining some relevant information.

- For each h^i, the measurements collected in each sub-experiment allow obtaining a grouped sample, that is $\{h^i\}$, which includes all measurements of h^i across all sub-experiments.
- For each h^i, the values of $\hat{\mu}_i$ and $\hat{\sigma}_i$ obtained from $\{h^i\}$.
- For each pair of HEMs h^i and h^j, we obtain $\hat{\rho}_{ij}$, which is the empirical counterpart of ρ_{ij}. $\hat{\rho}_{ij}$ is obtained from the sub-experiment where both h^i and $h^j$are observed together.

Therefore, since we have $\hat{\sigma}_i$, $\hat{\sigma}_j$ and $\hat{\rho}_{ij}$, we can also obtain their empirical covariance $\hat{\sigma}_{ij}$. Then, we can produce the empirical correlation matrix $\hat{S} \sim S$, and the corresponding (empirical) covariance matrix $\hat{\Sigma}$. Once we have constructed this information, we have a model for how each HEM in the experiment relates to all other HEMs in terms of expected values. Since we have $\hat{\mu}$ and $\hat{\Sigma}$ for all HEMs, we can describe the corresponding MVGD as follows:

$$X \sim \mathcal{N}_{nh}(\hat{\mu}, \hat{\Sigma})$$

However, now we need to generate actual merged HEM vectors using measured data in accordance with the MVGD model. From a mathematical point of view, the challenge is to reorder the grouped sample $\{h^i\}$ for each HEM h^i, such that for each pair of HEMs h^i and h^j the empirical correlation of the grouped samples is close to ρ_{ij}. Equivalently, the challenge is to reorder the grouped samples such that the new corresponding (empirical) covariance matrix is close to $\hat{\Sigma}$. This corresponds to an optimization problem with an existing solution, since the set of potential orders (all permutations)

is large but finite. However, it is not a trivial problem from a computational point of view. We construct the solution with a probabilistic approach using the preliminary results of copula theory [26].

Application of copula theory. For each h^i, the empirical distribution function F^i_{emp} lets us transform the grouped sample into a uniform sample. A uniform sample can be transformed into a Gaussian sample by applying the inverse function of the cumulative distribution function of a standard Gaussian distribution, Φ. Therefore, the grouped sample is transformed one-to-one into a standard Gaussian distribution:

$$\Phi^{-1}\Big(F^i_{emp}(\{h^i\})\Big) \tag{1}$$

For instance, if $\{h^i\}$ is $\{1, 2, 5, 8\}$, this would map to a uniform sample $\{0.2, 0.4, 0.6, 0.8\}$[1]. Then, those values would map to a standard Gaussian distribution as $\{-0.842, -0.253, 0.253, 0.842\}$[2].

We apply this process to all HEMs, thus obtaining for each measurement of each HEM its counterpart value for the standard Gaussian distribution. Then, we refer to as $\hat{\Sigma}_0$ to the covariance matrix obtained from the transformed data to differentiate it from the one obtained from the measured data ($\hat{\Sigma}$).

Finally, we generate a sample $samp_{MVGD}$ of n_{MVGD} runs (e.g. $n_{MVGD} = 10\,000$) from the MVGD, which we define using $\hat{\Sigma}_0$:

$$X \sim \mathcal{N}_{nh}(0, \hat{\Sigma}_0) \tag{2}$$

which produces a joint sample with marginal standard Gaussian distribution (i.e. sampled values follow such distribution). At this point, $samp_{MVGD}$ provides a matrix with as many columns as HEMs (nh), as many rows as HEM vectors we want to generate (e.g. as many as total measurements per HEM), and preserving the correlations across all pairs of HEMs (ρ_{ij}) simultaneously. However, values in the matrix correspond to a standard Gaussian distribution instead of being HEM values read. Thus, we use the actual $samp_{MVGD}$ to produce the indexes for order statistics. In other words, if for a given HEM h^i, $samp_{MVGD}$ has a particular order of values (e.g. kth lowest first, lth lowest second, mth lowest third, and so on and so forth), we set the actual observed values for h^i in the very same order to generate the merged HEM vectors. The easiest way to do this is setting n_{MVGD} to the actual number of measured values per HEM. For instance, if for a given HEM h^i the $samp_{MVGD}$ has produced the values $\{1.121, -0.870, -0.172, 0.343\}$, and the actual values observed are 9, 10, 12, 17, they would be sorted as follows: $\{17, 9, 10, 12\}$, thus preserving the same ordering, but this time using the actual values read. By following their $samp_{MVGD}$ ordering for all HEMs to organize the actual values measured, we generate as many merged HEM vectors as actual values have been observed for each HEM. For instance, recalling the example in Figure 4, where we have 12 values per HEM, we could set $n_{MVGD} = 12$, and would sort the values for each of the 7 HEMs in the same order as their synthetic values in $samp_{MVGD}$.

In fact, once this process is complete, we could assess $\hat{\rho}'_{ij}$ for all pairs of HEMs in the merged vectors and compare them with the original values $\hat{\rho}_{ij}$ obtained from pairwise HEM measurements. Some (small) discrepancy is expected due to statistical reasons (i.e. sampling processes can always produce inaccuracies). Such discrepancy could be reduced with an iterative process where X in Equation 2 is obtained as many times as needed and measured HEM values sorted accordingly to obtain new merged vectors where $\hat{\rho}''_{ij}$ is compared to $\hat{\rho}_{ij}$. This process could be repeated a fixed number of times or until a specific criterion is fulfilled. However, this step is purely optional.

Recalling the example for HRM limitations before, MUCH would successfully preserve all correlations across the three HEMs simultaneously, overcoming the limitation of HRM. In particular, MUCH does not favor any particular random variable (HEM) when merging and, instead, all correlations are preserved as accurately as possible at the same time. Instead, HRM strategy is a supervised one where correlations with a particular variable (anchor HEM) are perfectly preserved at the expense of causing large inaccuracies for other correlations if they are weakly correlated with the anchor.

4.4 Procedure

For the sake of completion, we provide the application process of MUCH, which includes five main steps. The procedure can be followed visually on Figure 3.

- **STEP** ①. The HEM selection for each sub-experiment does not play a role in MUCH. Each HEM will be measured with every other HEM at least once, to capture the relation between them, i.e. to compute the empirical correlation matrix $\hat{S}$. While generating the minimum number of sub-experiments allowing to capture all pairwise HEM correlations is convenient, it is not strictly mandatory for the application of MUCH, so combinations can be generated with greedy algorithms if needed. For each sub-experiment at least 30 runs ($nr \geq 30$) are needed to allow the use of the Central Limit Theory [18]. In general, the higher nr, the more accurate $\hat{S}$ will be. As a matter of fact, in this paper we set $nr = 50$.
- **STEP** ②. Once the values are gathered, map them to a standard MVGD, and compute the covariance matrix $\hat{\Sigma}_0$.
- **STEP** ③. Compute the MVGD using $\hat{\Sigma}_0$ as shown in Equation 2, and generate a sample equal in size (n_{MVGD}) to the number of collected values for each HEM in a sub-experiment or in all sub-experiments. Note that the method could also be applied with larger n_{MVGD} values.
- **STEP** ④. (OPTIONAL) As an optimization step, we can compute the correlation matrix of the generated sample $\hat{S}'$ from the MVGD and compare it to the measured correlation matrix $\hat{S}$, for instance, obtaining the Minimum Square Error (MSE). Then, we can repeat step ③ and keep the $\hat{S}'$ with lowest MSE compared to $\hat{S}$. Such process can be repeated as many times as wanted as a way to further increase accuracy without requiring additional runs on the target platform.
- **STEP** ⑤. Copy the order statistics of the sample of the MVGD into the experimental data. Now, the experimental data is finally merged in accordance with the MVGD.

[1] Given a HEM h^i for which we have n values, the uniform sample probability space is split into $n + 1$ identical parts. Out of the $n + 2$ boundary values, we exclude 0 and 1 since they cannot be used later for the Gaussian distribution as their counterpart values would be $-\infty$ and $+\infty$ respectively.

[2] As for the uniform distribution, those values distribute the probability space into $n + 1$ parts with identical cumulated probability.

Table 2: Workloads.

	Core 1	Core 2	Core 3	Core 4
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

Table 3: HEMs with observed relevant variability.

Name	HRM id
PROCESSOR_CYCLES	h^a
CYCLES_LSU_SCHE_STALLED	1
CYCLES_LSU_ISSUE_STALLED	1
BLINK_REQUEST	1
L2_MISSES	1
L2_DEMAND_ACCESSES	1
L2_ACCESSES	2
L2_STORE_ALLOCATES	2
L2_DATA_MISSES	2
L2_RELOADS_FROM_CORENET	2
L2_SNOOP_HITS	2
L2_SNOOP_PUSHES	3
STALL_FOR_RLT_CYCLES	3
STALL_FOR_WDB_CYCLES	3
BIU_MASTER_REQUESTS	3
BIU_GLOBAL_REQUESTS	3

5 EVALUATION

This section presents the experimental framework, the validation approach followed to evaluate MUCH and compare it with HRM, and the results of the evaluation.

5.1 Experimental Setup

We use an NXP T2080 MPSoC [12, 13], relevant for critical avionics systems [33], which includes 4 cores with private first level data (DL1) and instruction (IL1) caches per core, a unified second level (UL2) cache shared across cores, and an interconnect network to reach the memory controller, as well as a number of peripherals and interfaces (DMA, PCIe, SATA, DUART, Ethernet, etc.).

We run our tests on bare-metal and configure and read PMCs without using any library to minimize unwanted variability. The task running in *core 1* is the one for which we read the HEMs and apply our technique, although the very same process could be applied to any other task analogously. Benchmarks run with the same input data so they execute always the same instructions. We also run experiments in bare metal and reset the state of caches and TLBs across runs. Both help reducing any source of variability other than platform latent noise.

Our benchmarks have been devised to trigger a wide set of HEMs. In particular, they perform some core and memory operations with a variety of patterns, which include the operand type (Integer, Long, or Floating point), the type of core operation (ADDition or MULtiplication), and the level of the memory hierarchy accessed (DL1, UL2, or main MEMory), thus leading to 18 different combinations. For instance, LMUL_UL2 performs long integer multiplications with data fetched mostly from UL2. For our evaluation, we have generated the 16 workloads shown in Table 2.

5.2 Validation Approach

The reference against which to compare MUCH is the actual correlation of each pair of HEMs when measured together in the platform, so that we can validate whether correlations in the merged HEM vector are accurate w.r.t. real correlations. For completeness, we compare MUCH against HRM in terms of both, accuracy w.r.t. the real correlations and number of runs required.

For the sake of comparison, we focus on the same 16 HEMs regarded as relevant in HRM [39], which we list in Table 3 for completeness. Note that, while all HEMs are treated homogeneously by MUCH – thus meaning that all pairwise HEM correlations are measured and then processed together in an identical basis – the same does not apply to HRM. In particular, HRM needs a HEM to be the anchor. Then, given that the T2080 MPSoC has 6 PMCs and one of them is used by the anchor, the remaining 15 HEMs need to be distributed across 3 sub-experiments. The sub-experiment where each HEM is read for HRM is shown in Table 3 in the *HRM id* column.

Note that, by using 16 HEMs, there are 120 different pairs of HEMs for which we evaluate the actual correlation obtained for the merged HEM vectors with MUCH and HRM, and compare them against their real empirical correlation, $\hat{\rho}^{i,j}$. In particular, we compute the accuracy for both methods as $\left|\rho^{i,j}_{much} - \hat{\rho}^{i,j}\right|$ and $\left|\rho^{i,j}_{hrm} - \hat{\rho}^{i,j}\right|$.

5.3 Results

We evaluate the correlation accuracy obtained, for both MUCH and HRM, against the real empirical correlation. For this first comparison, we set $nr = 50$. The number of sub-experiments is only 3 for HRM ($nb = 3$). For MUCH, while theoretically we could observe 120 pairs of HEMs with $nb = 8$ sub-experiments (15 pairs per sub-experiment with 6 PMCs), we needed $nb = 10$ just following a greedy process to create sub-experiments where we iterate over HEMs from 1 to 16, and for each one we create sub-experiments adding the lowest order HEM not yet observed with any of already selected HEMs in the sub-experiment. Therefore, $N = 150$ for HRM and $N = 1000$ for MUCH.

We show detailed results for the 120 pairs of HEMs in Figure 5 for workloads W1, W2, W3, and W5. In particular, W2 corresponds to the case where the improvement of MUCH w.r.t. HRM is only moderate, W3 to an extreme case with huge improvement, and W1 and W5 to two cases with typical high improvement. The HEM pair values are sorted from lowest difference to highest difference w.r.t. the real correlation for each technique. As shown, MUCH provides higher accuracy since its differences w.r.t. the real correlation are

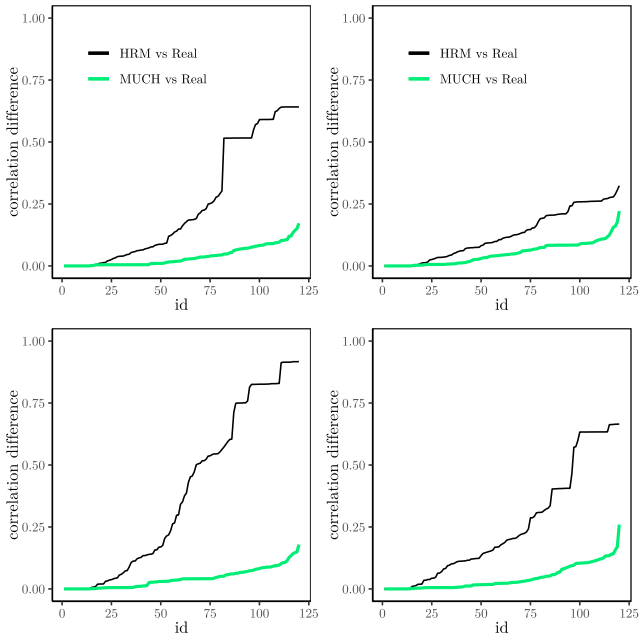

Figure 5: Correlation Difference for each HEMs pair for workload 1 (top-left), workload 2 (top-right), workload 3 (bottom-left), and workload 5 (bottom-right).

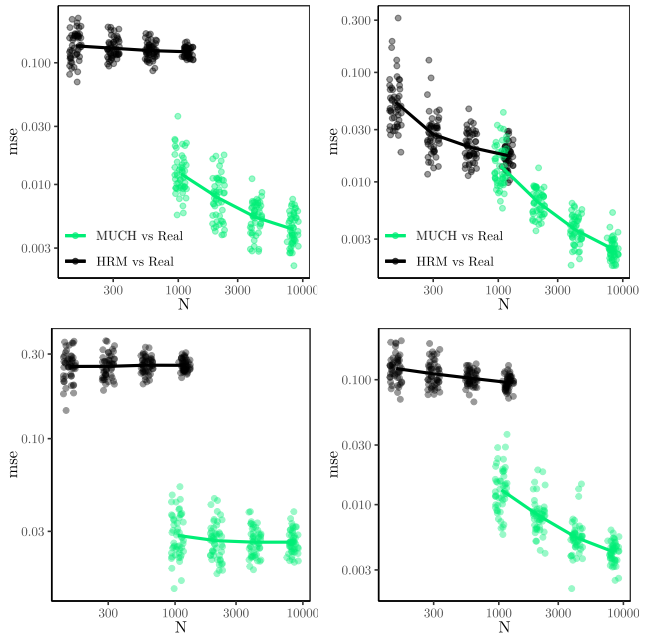

Figure 6: Mean square error for HRM and MUCH as a function of the number of runs. Workload 1 (top-left), workload 2 (top-right), workload 3 (bottom-left), and workload 5 (bottom-right).

much lower than those of HRM. Moreover, the difference in the worst case for MUCH is up to 0.25 for very few pairs of HEMs, whereas HRM reaches values above 0.5 for a non-negligible number of pairs, and even above 0.75 in some cases. Note that the maximum theoretical difference is 2.0, which would occur when the estimated correlation is 1.0 (or -1.0), and the real correlation is -1.0 (or 1.0).

As a second comparison, we study the dependence of each method on the total number of runs $N = nr \cdot nb$, which illustrates the trade-off between cost (in terms of number of runs) and accuracy for both methods. Again, we consider the same 4 workloads, where we vary nr (values 50, 100, 200 and 400), and obtain for each workload, the MSE for their difference w.r.t. the real correlation across the 120 pairs of HEMs. In particular, to produce a statistically significant comparison, we bootstrapped 50 samples for each of the methods and each nr value. For instance, for $nr = 100$ this implies that we generate a random sample of 100 runs for each sub-experiment and apply the corresponding method on that sample. Then, we repeat the process 50 times, thus obtaining 50 estimates for each method and nr value.

Figure 6 shows those results, where dots indicate individual measurements and the line corresponds to the mean across them. Note that both axes are in logarithmic scale. First, we observe that HRM obtains negligible gains from increasing N. Those gains are only noticeable for W2, where pairwise correlations with the anchor HEM are indeed high, allowing HRM to be almost as accurate as MUCH. In any case, HRM apparently plateaus at $N = 1200$ ($nr = 400$).

For MUCH, we observe significant gains in all cases but W3, where increasing N produces limited improvements in accuracy. However, in the other 3 cases we observe improved accuracy as we increase N, and, apparently, such improvement does not plateau even with $nr = 400$, thus offering opportunities to further increase accuracy if the number of runs is increased beyond that number.

When comparing MUCH and HRM, we note that in general, MUCH allows reaching much more accurate merged HEM vectors. It is of prominent importance the case where $N \approx 1000$, because it allows performing an iso-cost comparison across both methods, i.e. with the same number of total runs. In this case, where $N_{much} = 1000$ ($nr_{much} = 50$) and $N_{hrm} = 1200$ ($nr_{hrm} = 400$), thus with a slight advantage for HRM, we observe that MUCH is significantly better than HRM in 3 out of 4 workloads, and in the remaining one, where the MSE is already pretty low for both methods, MUCH is slightly better despite its slightly lower number of runs. Finally, note that in both methods, increasing N reduces dispersion of the bootstrap, thus making merged HEM vectors more stable in terms of accuracy w.r.t. the real correlations.

For completion, we have evaluated all workloads with $N = 2100$, so $nr_{hrm} = 700$ and $nr_{much} = 210$, again with a bootstrap with 50 samples. The mean MSE across the 50 samples is shown in Table 4. As expected, MUCH is systematically more accurate than HRM for all workloads, and the difference across both methods is tiny (e.g. W2 and W7) only when HRM is highly accurate, since MUCH is always highly accurate. In fact, the worst MSE for MUCH (0.020 for W9) is indeed better than the best MSE for HRM (0.021 for W7).

So far we have shown that MUCH outperforms HRM under iso-cost (identical number of runs N), and naturally, under identical number of runs per sub-experiment (iso-nr), where MUCH has a

Table 4: MSE for the 16 workloads under iso-runs ($N = 2100$).

Workload	MUCH	HRM
W1	0.003	0.135
W2	0.004	0.024
W3	0.004	0.295
W4	0.006	0.272
W5	0.005	0.110
W6	0.007	0.083
W7	0.006	0.021
W8	0.006	0.068
W9	0.020	0.228
W10	0.007	0.168
W11	0.017	0.207
W12	0.005	0.158
W13	0.008	0.282
W14	0.005	0.106
W15	0.007	0.068
W16	0.017	0.250

higher N value. Moreover, we have shown that in all cases MUCH is highly accurate. However, the number of sub-experiments needed by MUCH is much more dependent on nh and np than that of HRM. For instance, if nh is high or np is very low, MUCH may need many sub-experiments whereas HRM only needs $N_{hrm} = nr \cdot \lceil \frac{nh-1}{np-1} \rceil$. In particular, for MUCH we need to observe $\binom{nh}{2} = \frac{nh \cdot (nh-1)}{2}$ pairs of HEMs, and each sub-experiment allows observing up to $\binom{np}{2} = \frac{np \cdot (np-1)}{2}$ pairs. Assuming that sub-experiments for MUCH can be optimized to generate always unobserved pairs of HEMs only, the number of sub-experiments would be ratio between both values, and hence, the total number of runs would be $N_{much} = nr \cdot \lceil \frac{nh \cdot (nh-1)}{np \cdot (np-1)} \rceil$. Figure 7 shows N_{much} and N_{hrm} for the case $np = 6$, as in the T2080, and $nr = 100$, when varying nh. As shown, N_{much} grows much faster than N_{hrm} as we increase nh. Therefore, there may be cases where N_{much} might not be affordable and the only affordable solution is HRM. In those cases, despite the limitations of HRM, such solution has been shown to be systematically better than any other alternative (except MUCH) [39], and thus, it would be the best choice.

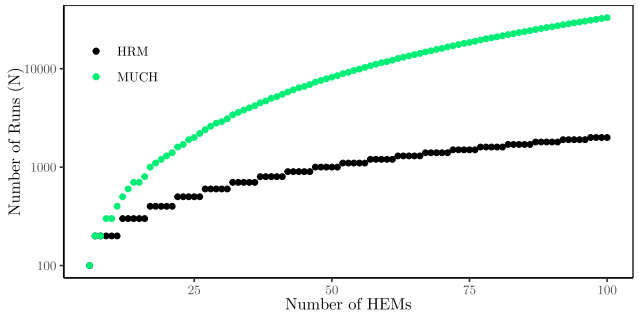

Figure 7: Number of total runs as a function of the number of HEMs to arrange. The parameters on N_{much} and N_{hrm} are $np = 6$, $nr = 100$.

6 RELATED WORK

Several solutions have been proposed to master multicore interference in industrial contexts. Some approaches attempt to reduce or fully remove interference across tasks running concurrently in multicores by segregating accesses to different hardware blocks. These approaches have been devised for on-chip and off-chip memories, including banks of shared caches, as well as banks and ranks of DDR memories [16, 20–22, 29, 37]. Other approaches perform segregation over time, rather than over space, by splitting execution of tasks into memory and computation phases, thus letting schedulers guarantee that memory phases from different tasks do not occur simultaneously [3, 6, 31]. However, segregation over time is not always doable due to the characteristics of the hardware or the application itself (e.g. application semantics cannot be changed due to overwhelming verification and validation costs). Therefore, if time segregation is not feasible, even if space segregation is used, multicore interference can still occur in hardware resources not visible at software level, such as interconnects, as well as buffers and queues internal to shared caches for instance [28]. As discussed before, this is the case for the NXP T2080 considered for critical avionics systems [33]. In those cases, solutions are needed to master interference in multicores and account for it during timing analysis.

Several works build on HEMs for multicore timing analysis, for bound estimation and online monitoring [7, 14, 15, 27]. On the industrial side, HEMs have also been exploited to produce timing evidence needed for certification on multicores [33, 38].

Several works in the high-performance domain study the source of HEM values variability. They found some sources in software layers such as the operating system, the application itself or the HEM interface library [1, 24, 25, 44]. However, none of those applies in our context where we exclude them by exercising sufficient control in the system, much in line with critical real-time embedded systems practices. Sources at hardware level have already been identified in the form of hypotheses [40]. However, despite they can be identified in some cases, explicit control cannot be exercised and variability exists anyway.

Alternative approaches to merging HEMs could be considered, such as for instance, Matrix Completion methods [17, 34]. Unfortunately, Matrix Completion requires particular properties for the matrix that HEMs values cannot fulfill, such as being a random matrix with rows and columns belonging to the same distribution. Such property cannot be met because, for instance, the distribution of one HEM (e.g. processor cycles) is different to that of any other HEM (e.g. L2 cache misses), since they have different mean, variance and shape.

Finally, to our knowledge the only work tackling the same problem as MUCH is HRM [39], whose pros and cons have been deeply discussed all along the paper, and compared against MUCH proving that MUCH merges HEM vectors with much higher accuracy w.r.t. real observed data.

7 CONCLUSIONS

Multicore timing analysis often builds upon HEMs to produce reliable execution time bounds for critical real-time applications in avionics, automotive and space domains among others. However, the number of PMCs is too small to allow reading all relevant HEMs

at once, and variability across runs is unavoidable, so end users need solutions to merge HEM values into consistent vectors resembling the case where they had been read all together. So far, only HRM has been proposed to tackle this problem. While its requirements in terms of number of test runs needed is low, the quality of the merged HEM vectors by HRM is limited due to its inability to preserve correlations among many pairs of HEMs. This paper presents MUCH, which overcomes HRM limitations by smartly observing pairwise HEM correlations and building on multivariate Gaussian distributions to generate merged HEM vectors where discrepancies between produced pairwise HEM correlations and real ones is very low. Our results on an NXP T2080 MPSoC used for avionics critical systems support the effectiveness of MUCH and prove that it outperforms HRM systematically, thus being HRM appropriate only in those cases where too many HEMs are needed and there are very few PMCs.

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] A. R. Alameldeen and D. A. Wood. 2003. Variability in architectural simulations of multi-threaded workloads.
[2] A. Betts et al. 2010. Hybrid measurement-based WCET analysis at the source level using object-level traces. In *WCET workshop*.
[3] A. Biondi and M. Di Natale. 2018. Achieving Predictable Multicore Execution of Automotive Applications Using the LET Paradigm. In *RTAS*.
[4] J. Cardona et al. 2019. Maximum-Contention Control Unit (MCCU): Resource Access Count and Contention Time Enforcement. *DATE* (2019).
[5] D. Dasari et al. 2011. Response Time Analysis of COTS-Based Multicores Considering the Contention on the Shared Memory Bus. In *TrustCom*.
[6] DDC-I. 2020. Patent Details for Managing Cache. https://www.ddci.com/manage_cache_patent/.
[7] E. Díaz et al. 2018. Modelling multicore contention on the AURIX TC27x. In *DAC*.
[8] E. Díaz et al. 2018. Modelling multicore contention on the AURIX^{TM} TC27x. In *DAC*.
[9] B. Dreyer et al. 2016. Continuous Non-Intrusive Hybrid WCET Estimation Using Waypoint Graphs. In *WCET workshop*.
[10] G. Fernandez et al. 2014. Contention in Multicore Hardware Shared Resources: Understanding of the State of the Art. In *WCET Workshop*.
[11] G. Fernandez et al. 2017. Computing Safe Contention Bounds for Multicore Resources with Round-Robin and FIFO Arbitration. *IEEE Trans. Comput.* 66, 4 (April 2017), 586–600. https://doi.org/10.1109/TC.2016.2616307
[12] Freescale semicondutor. 2014. e6500 Core Reference Manual. https://www.nxp.com/docs/en/reference-manual/E6500RM.pdf. E6500RM. Rev 0. 06/2014.
[13] Freescale semicondutor. 2016. QorIQ T2080 Reference Manual. Also supports T2081. Document Number: T2080RM. Rev. 3, 11/2016.
[14] D. Griffin et al. 2017. Forecast-based Interference: Modelling Multicore Interference from Observable Factors. In *RTNS*.
[15] F. Guet et al. 2016. Probabilistic analysis of cache memories and cache memories impacts on multi-core embedded systems. In *SIES*.
[16] H. Yun et al. 2014. PALLOC: DRAM bank-aware memory allocator for performance isolation on multicore platforms. In *RTAS*.
[17] T. Hastie et al. 2015. Matrix Completion and Low-Rank SVD via Fast Alternating Least Squares. *J. Mach. Learn. Res.* 16, 1 (Jan. 2015).
[18] R. V. Hogg and E. A. Tanis. 1997. *Probability and statistical inference.* Prentice Hall.
[19] J. Jalle et al. 2013. Deconstructing bus access control policies for Real-Time multicores. In *SIES*.
[20] T. Kloda et al. 2019. Deterministic Memory Hierarchy and Virtualization for Modern Multi-Core Embedded Systems. In *RTAS*. IEEE.
[21] L. Liu et al. 2012. A software memory partition approach for eliminating bank-level interference in multicore systems. In *PACT*.
[22] R. Mancuso et al. 2013. Real-time cache management framework for multi-core architectures. In *RTAS*.
[23] E. Mezzetti, L. Kosmidis, J. Abella, and F. J. Cazorla. 2018. High-Integrity Performance Monitoring Units in Automotive Chips for Reliable Timing V V. *IEEE Micro* 38, 1 (2018), 56–65.
[24] T. Mytkowicz et al. 2009. Producing wrong data without doing anything obviously wrong!. In *ASPLOS*.
[25] R. Neill et al. 2017. Fuse: Accurate Multiplexing of Hardware Performance Counters Across Executions. *TACO* 14, 4, Article 43 (2017).
[26] R. B. Nelsen. 2010. *An Introduction to Copulas.* Springer Publishing Company, Incorporated.
[27] J. Nowotsch et al. 2014. Multi-core Interference-Sensitive WCET Analysis Leveraging Runtime Resource Capacity Enforcement. In *ECRTS*.
[28] P. K. Valsan et al. 2016. Taming Non-Blocking Caches to Improve Isolation in Multicore Real-Time Systems. In *RTAS*. 161–172.
[29] X. Pan and F. Mueller. 2018. Controller-aware memory coloring for multicore real-time systems. In *SAC*. ACM, 584–592.
[30] R. Pellizzoni et al. 2010. Worst case delay analysis for memory interference in multicore systems. In *DATE*.
[31] R. Pellizzoni et al. 2011. A Predictable Execution Model for COTS-Based Embedded Systems. In *RTAS*.
[32] J. Pérez-Cerrolaza et al. 2020. Multi-core Devices for Safety-critical Systems: A Survey. *ACM Comput. Surv.* 53, 4 (2020).
[33] D. Radack et al. (Rockwell Collins). 2018. Civil Certification of Multi-core Processing Systems in Commercial Avionics.
[34] B. Recht. 2011. A Simpler Approach to Matrix Completion. *J. Mach. Learn. Res.* 12, null (Dec. 2011), 3413–3430.
[35] J. Reineke. 2017. Challenges for Timing Analysis of Multi-Core Architectures. Workshop on Foundational and Practical Aspects of Resource Analysis. Invited Talk.
[36] K. Schmidt et al. 2015. Non-Intrusive Tracing at First Instruction, In SAE Technical Paper. https://doi.org/10.4271/2015-01-0176
[37] N. Suzuki et al. 2013. Coordinated Bank and Cache Coloring for Temporal Protection of Memory Accesses. In *CSE*.
[38] S. H. VanderLeest and C. Evripidou. 2020. An Approach to Verification of Interference Concerns for Multicore Systems (CAST-32A), In SAE Technical Paper. https://doi.org/10.4271/2020-01-0016
[39] S. Vilardell, I. Serra, R. Santalla, E. Mezzetti, J. Abella, and F. J. Cazorla. 2020. HRM: merging hardware event monitors for improved timing analysis of complex MPSoCs. *IEEE transactions on computer-aided design of integrated circuits and systems* (Sep 2020). http://hdl.handle.net/2117/327538
[40] V. M. Weaver et al. 2013. Non-determinism and overcount on modern hardware performance counter implementations. In *ISPASS*.
[41] R. Wilhelm and J. Reineke. 2012. Embedded systems: Many cores - Many problems. In *SIES*.
[42] S. N. Wood. 2015. *Core Statistics.* Cambridge University Press.
[43] H. Yun et al. 2013. MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In *RTAS*.
[44] D. Zaparanuks et al. 2009. Accuracy of performance counter measurements. In *ISPASS*.