Forecast-Based Interference: Modelling Multicore Interference from Observable Factors

David Griffin
University of York
david.griffin@york.ac.uk

Benjamin Lesage
University of York
benjamin.lesage@york.ac.uk

Iain Bate
University of York
iain.bate@york.ac.uk

Frank Soboczenski
Kings College London
frank.soboczenski@kcl.ac.uk

Robert I. Davis
University of York
rob.davis@york.ac.uk

ABSTRACT

While there is significant interest in the use of COTS multicore platforms for Real-time Systems, there has been very little in terms of practical methods to calculate the interference multiplier (i.e. the increase in execution time due to interference) between tasks on such systems. COTS multicore platforms present two distinct challenges: firstly, the variable interference between tasks competing for shared resource such as cache, and secondly the complexity of the hardware mechanisms and policies used, which may result in a system which is very difficult if not impossible to analyse - assuming that the exact details of the hardware are even disclosed! This paper proposes a new technique, Forecast-Based Interference analysis, which mitigates both of these issues by combining measurement based techniques with statistical techniques and forecast modelling to enable the prediction of an interference multiplier for a given set of tasks, in an automated and reliable manner. The combination of execution times and interference multipliers can be used both in the design (e.g. for specifying timing watchdogs) and analysis (e.g. schedulability analysis) of systems.

1. INTRODUCTION

Two main drivers of change in computer systems are the movements towards faster and cheaper platforms. Partly driven by practical limits on processor clock speeds, there is a shift to multicore platforms, where two or more processor cores concurrently execute instructions while sharing a number of resources. The movement to multicore systems has resulted in a well know problem: interference between tasks. Interference between multiple cores occurs when they compete for the same resource. Interference is especially problematic as the amount of interference generated by a contender task may not be specified or may be highly variable, and may not be bounded. Hence there are cases where traditional techniques, such as measurement based analysis, are incapable of giving useful results. For example, in the development of a system not all tasks may be implemented, and as such measurement based analysis is incapable of giving any information about the effects of interference from currently unimplemented tasks. Therefore, in the case that due to the phased development of software not all contender tasks are available upon completion of the task under analysis, measurement based analysis is not useful which makes incremental timing verification impossible. This is important in that if testing could be carried out on tasks as they become available, these tests could determine probable bounds on their resource usage, and this in turn could be used to shape the development of the unavailable tasks and reduce the likelihood of problems once they are integrated into the complete system.

The movement towards COTS platforms presents an additional challenge: the surrender of control over hardware. When using specially designed real-time platforms, e.g. the avionic systems platform described by Law [13], a number of features could be implemented to aid the computation of timing constraints. As the name implies, COTS processors may not have these features, limiting the applicability of techniques such as leaky bucket throttling [11], and in some cases may implement schemes which are unknown due to manufacturers wishing to obscure the details of their platform from contenders for the sake of protecting their intellectual property. Therefore classes of techniques, such as static analysis [4], which rely on fully understanding the properties of the system under analysis become infeasible.

Further, even in the case where hardware does have the requisite features, the real-time properties of such schemes may not work as expected. One example of this is cache partitioning [23]. Cache partitioning is implemented by restricting the ability of concurrently running tasks to utilise the entire shared cache. As the tasks no longer contend for space in the cache, the interference is reduced. However, recently it has been shown by Farshchi et al. [18] that even with cache partitioning in place, it is still possible for cache related competition to occur, and for the effects of this competition to be significant. Indeed, Farshchi et al. showed that competition for cache miss status holding registers can cause a slowdown of up to twenty times, demonstrating that even when interference is mitigated there still exists the possibility of large and unpredictable effects from tasks running on other cores.

In light of these observations, there is a strong argument

RTNS '17, October 4–6, 2017, Grenoble, France

ACM ISBN 978-1-4503-5286-4/17/10…$15.00
https://doi.org/10.1145/3139258.3139275

that approaches reliant on either completely understanding or mitigating multicore interference are at best difficult to apply to COTS platforms, and at worst impossible. As it is not desirable to return to the higher costs and lower performance of specialist or customised real-time processors, a different approach is required to characterise multicore interference. At this stage, black box techniques, such as statistical methods [15] appear to be free of these limitations. Unfortunately, the black box techniques used to date tend to be univariate which leads to a significant limitation; whilst they may be able to determine if a property of a system holds, they do not work as well when the system is not completely defined. For example, in the implementation of a system, different teams may be tasked with implementing different subsystems. Each team of implementers will want to be able to test their component in isolation to ensure that it will work on the finished system, but black box testing is only capable of determining properties relating to the interference between tasks when all tasks are present. Therefore, it is impossible for each team of implementers to guarantee that their component will work in the finished system, which complicates development and testing. If these issues were addressed by some form of intermediate model, this opens the door to effective parallel development and compositional analysis, where different components developed in parallel have temporal properties that interact in a well-defined manner.

Find a reference for claim that Deep Learning may behave better than holistic approaches.

We conclude there are hard real-time systems for which there is no real method to understand their detailed timing behaviour. Instead multi-variate statistical verification that relate the inflation to the platform and software factors that cause it must be used to determine the validity of the system, and give guarantees about its timing properties. As the critical definition of real-time systems is that the timeliness of an answer impacts the correctness of a system, one can argue that it is not necessary to fully understand the method by which a worst-case execution time estimate is obtained, provided that sufficient statistical testing can be carried out to validate its accuracy. This opens up the possibility that Deep Learning algorithms can be used to compute components of the execution time, such as multicore interference, where the state of the art is unable to do so using traditional methods. The key difference to more traditional neural networks is that Deep Learning uses many more layers that allow the resulting network to represent the systems in a hierarchical fashion. From a timing perspective, this means the network can automatically separately learn the behaviour of the system when significant effects occur (e.g. preemption from a new task, effectively resulting in a cache flush) and then for each of these significant effects a more detailed model is separately established. The overall network is an ensemble of the collection of effects (significant or otherwise). The Deep Learning's hierarchy in effect gives a divide and conquer approach allowing detailed models to be established and learned without confusion from un-related events. For complex problems this has proved to be more effective than an holistic approach.

1.1 Contribution

To take advantage of the observation that deep-learning with statistical guarantees is appropriate for a number of hard real-time systems, a new type of analysis has been constructed. The purpose of this analysis is not to determine the single-core Worst-Case Execution Time (WCET) estimate for a task as if it has been executed in isolation as this can be calculated using existing techniques such as those based on search-based techniques [13], Extreme Value Theory [15], and commercial tools such as Rapitime [22]. Instead this paper is specifically concerned with calculating how the execution time of the task may be inflated by interferences from tasks sharing resources in a multi-core context. The intention is that the analysis provides a parameterised model of the interference so that a worst-case interference multiplier can be determined without restrictions on resource usage, the interference multiplier can be tightened if the sharing of resources is controlled (e.g. a limited number of accesses to a shared bus by each task is enforced), and where tasks do have restrictions on their use of shared resources then their usage can be verified as appropriate.

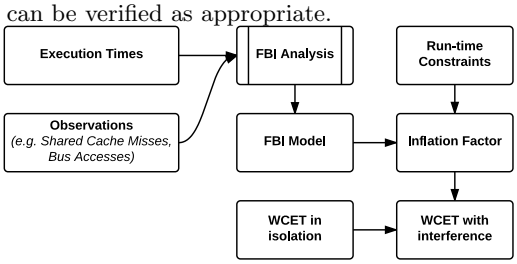

Figure 1: Overview of FBI Analysis

Illustrated in Figure 1, the new technique, Forecast Based Interference (FBI) analysis, takes both execution times and observations made on the task under analysis. The observations are used to characterise the interference from a set of synthetic contender tasks. By applying techniques from forecasting [16] and deep-learning [1, 6], FBI analysis constructs an FBI model of the task which provides a mapping between observed rates of interference and the effect of that interference on the execution time of the task under analysis. The resulting FBI model can be queried with the constraints given for the system as it will be deployed to find an interference multiplier. As deep-learning is used to construct the model, no attempt is made to understand how the model works; instead, statistical testing on an unseen data set is used to determine the accuracy of the model to the required level of confidence. Finally, the interference multiplier can be applied to a single-core WCET estimate (obtained by any appropriate method) to derive a multicore WCET estimate for the given level of interference. To the best of our knowledge, the construction and deployment of such a model has not been accomplished before. The primary advantage of FBI analysis is that it makes few assumptions about processor features or behaviours, instead relying only on features such as Performance Monitoring Counters (PMC) commonly found in COTS hardware.

In addition to the usage outline above, the FBI model is versatile enough to support alternative uses. For example, one can use the FBI model to find the maximum possible interference that can be observed (by means of search on the model). A second example is the use of the FBI model to facilitate parallel or incremental development by including the maximum level of interference a subsystem may generate as part of the specification of the subsystem. In this case, even though there may be no available observations for the actual co-running tasks, the model is

capable of giving a prediction about their effects. If hardware or software support is available, these bounds could also be enforced by an appropriate control mechanism (e.g. throttling), which would give further confidence in the results at the expense of additional overheads.

1.2 Related Work

Radojković et al. [21] provide an empirical evaluation of the effects of interference from contender tasks. While the COTS processor chosen is not common for real-time applications, they demonstrate that there are a wide variety of factors which can have a substantial impact on the execution time of a task. This confirms that any WCET analysis of a multicore system must take into account any factors which have not been controlled for. Recent work by Yun et al. [26] has provided an accurate, multicore model of the memory subsystem of a COTS processor. In addition to demonstrating the complexity of COTS hardware, this work also shows that there is a significant gap between the theoretical worst case performance of a system and what can be observed given a set of contender tasks. This highlights the fact that the theoretical worst case behaviour of a system may not match the worst case behaviour given a set of contender tasks.

Forecasting [16] is the name given to the family of techniques used to predict information about events which have not yet been observed. There are many different types of forecasting, but all rely on the same principle: constructing a *forecast model* of the system under study which can be used to predict how the system will behave under unobserved conditions. In the field of real-time systems, Zheng [27] applied linear regression techniques to relate the amount of resource accesses (obtained via the PMCs) to the inflation in the execution times caused by the accesses, however in our experience a linear relationship does not hold for most tasks and platforms. Griffin et al. [7] employed forecasting to determine information about the behaviour of tasks when their execution time budgets were exceeded, by constructing a model based on the observed behaviour of the tasks execution times. While Griffin et al.'s work focused on the technique of extrapolation, this paper employs Neural Networks [17], a machine learning approach capable of learning sophisticated patterns in data and making predictions based on these learned patterns. One of the benefits of this approach is that it is more capable of handling multivariate models, as opposed to extrapolation which is better suited to univariate problems.

Multicore interference and its effect on the WCET of a task has been explored in work by Paolieri et al. [20] who describe the IA^3 algorithm. IA^3 is an interference aware multicore resource allocation algorithm which allows for each task to have multiple WCETs, depending on the amount of interference generated by co-running tasks. In evaluating their algorithm, Paolieri et al.'s attempt to find the worst case interference by using synthetic contenders which access shared resources as frequently as possible. While on the platform used [19] this is a reasonable assumption, it leaves open the question of how the approach can be extended to COTS platforms where the worst case interference is non-trivial to find.

1.3 Structure

Section 2 provides detail on the new approach, FBI analysis. This is split into subsections concerned with data collection 2.2, a description of the hardware platform used in this paper 2.3, how the amount of data required is reduced to a manageable amount 2.4, how the model is constructed 2.5 and how trust can be established 2.6. This culminates in the overall algorithm in Section 2.7. An evaluation on the test platform is carried out in Section 3, which includes publicly available synthetic benchmarks as well as an industrial case study. Finally, conclusions are drawn in Section 4.

2. FBI ANALYSIS

The goal of FBI analysis is to create a black box model which can map between a set of easily observable PMCs and their effect on the interference suffered by a task. As path data is not used, the model must be capable of handling information from various paths through the task. The length of an execution path impacts the number of observed interference events during execution, whereas event rates are comparable even if the execution time of distinct paths differs. Hence, the model is described as taking the rate at which the observed PMCs change and find an interference multiplier which can be applied to a single core execution time to produce a multicore execution time for the given rates of interference. However the model is constructed, it must be able to handle complex features in the data set. If the model is too simple, it will not be able to handle features such as discontinuities in the effects of interference [17]. These effects are likely to be caused by how different paths of a task react to interference, and are thus relatively common.

As with any measurement-based technique, the execution time of a job is a critical piece of information to gather. However, it is also required that observations are made which capture data on the events that occur on the cores of the platform during the job's execution; for this work we utilise the PMCs which most modern processors possess [9]. While PMCs are primarily used in applications such as compiler optimisation, the information they expose can be used to characterise the usage of shared resource [5]. For example, PMCs indicating cache misses signify an access to main memory over the shared bus. As each core is able to write to its own PMCs, it is possible to use PMCs to obtain a characterisation of each core's usage of shared resources. Using PMCs in this way does present some problems however: there are typically more PMCs available than physical registers in which they can be stored [9], and so the selection of PMCs is critical. If the PMCs being used for the analysis do not correspond to useful interference effects then any analysis based on these metrics would be inaccurate (either pessimistic or optimistic). Further, while it can be inferred that there is some relation between the PMCs and the execution time of the analysed tasks, for this to be useful an analysis technique based on these PMCs must define this relationship.

2.1 Overview

FBI analysis consists of the following five principal stages. The first two of these phases have already been presented in [14][1]. In [14], detailed information on how the contenders are actually implemented are given as well as an evaluation of their effectiveness in testing the system

[1]A pre-release of this paper can be found at http://www.cs.york.ac.uk/∼ijb/automotive_2017.pdf.

and the appropriateness of the PMCs selected.

1. Initial Data Collection: Data collection is carried out by testing the task of interest on a multicore platform against contender tasks and capturing as many performance counters as possible in addition to execution times. All measurements are made as end-to-end measurements; there is no requirement to collect data at any intermediate point during execution. Further details on this step are given in Section 2.2.

2. PMC Selection: While in traditional machine learning approaches as much data as possible would be used, the practical reality is the available PMCs are limited and repeating tests to capture the effects of a wider range of PMCs would be expensive. Therefore, the Principal Component Analysis (PCA) technique is applied to identify correlated PMCs along the axes of the Principal Components (PCs) of the gathered data. Using the information in the principal components, a set of PMCs' execution time variations is chosen. Details on this step are given in Section 2.4.

3. Main Data Collection: Once the representative PMCs have been identified, the majority of data collection is carried out only collecting data on PMCs selected by PCA as well as execution times. Again, the measurements required are end-to-end measurements of the task running against contender tasks. As this is a restricted form of the initial data gathering step, the details are the same as given in Section 2.2.

4. Modelling: Using the data gathered in the main data collection, multiple forecast models are constructed using automated modelling to determine the effect of interferences on the task, which are expressed as a multiplier. As these models are constructed using machine learning, additional precautions have to be taken against accepting models which are only accurate on a portion of the domain; this is accomplished by the creation of multiple models which are used to implement Ensemble Modelling [17]. Additional details are given in Section 2.5

5. Trust: Once the model has been created, it is necessary to determine the trustworthiness of the model. This is accomplished by evaluating its forecast accuracy against unseen experimental data, which yields both a margin for error and the statistical confidence that this margin represents an upper bound on the interference effects. This is described in Section 2.6

Its important to note that every stage of the process is dependent on the task and platform being analysed, e.g. one task may make more use of data than another task which mainly accesses devices. Therefore all of the above five steps in the process have to be repeated for each combination of task under analysis and platform.

2.2 Data Collection

As the FBI approach relies on detailed task-level instrumentation, it is necessary to define the precise requirements. In addition to the execution time of the analysed task being necessary for the analysis, it is also necessary to collect other metrics in order to establish links between observed events and their impact on the execution time. In order to accomplish this, the PMCs which hardware platforms expose are utilised. PMCs allow for counts of specific events, e.g. cache misses or pipeline stalls, which can then be used as a proxy for the actual level of interference between tasks at a high level. For example, a high level of cache misses, but only when running against a task contending for the cache, is indicative of a high level of cache-related interference.

Initially, all PMCs must be captured. The platform used, the Infineon AURIX [10], is representative of the typical problem faced: the AURIX, like many real-world systems, is incapable of capturing all PMCs simultaneously as there are only limited registers available for PMC use. In order to combat this, tasks were run multiple times with the same inputs, capturing different PMCs on each run. As the traces for identical runs produce different results, due to interference from uncontrollable sources (e.g. physical instability in the chip, uninitialised values when the chip is powered on), it was necessary to find the level of error that this approach would introduce. This was accomplished by fixing a performance counter and comparing its values to when other performance counters were used; this demonstrated that the error introduced was minimal ($< 5\%$). Further the Wald-Wolfowitz [25] test was employed, which confirmed that the error observed could be reasonably characterised as random noise, and therefore the error would not introduce systemic failings [25]. In practice these tests only need to be carried out once per platform, and therefore once these properties have been verified instrumentation can be fully automated.

Data collection must observe the effects of competition from contending tasks. These contending tasks could be the actual tasks competing for resources when the system is deployed. However, for the reasons previously discussed, there are a number of situations where these tasks cannot be used (e.g. the tasks have not been developed yet). As such, FBI analysis uses synthetic contenders which are held in and executed from main memory. Unlike in previous work [20], where synthetic contenders were used to create an assumed worst-case scenario, the synthetic contenders used in FBI analysis must be able to exercise resources at varying rates of interference. The contenders are designed to exercise the shared resources over the range of values of interest, i.e. to systematically exercise the resource to give a good quality model. For example, a synthetic contender task may access shared memory every N cycles, and shared storage every M cycles. The values of M and N are stepped through the desired range of the model. We note that the resulting model is only valid for this range. In this paper, synthetic contenders are used which exercise shared memory at a controlled rate; as all tasks under consideration are run from the scratchpad with the generated data being stored in shared memory, this satisfies the condition that all sources of contention can be exercised at different interference levels. In our framework varying and exercising interferences relies on the synthetic contenders. In the absence of scratchpad, instruction memory layers, e.g. Flash or cache, could be additional sources of interference and would need to be accordingly observed under different interference configurations to apply the analysis. More details on the implementation of the contenders can be found in [14].

2.3 Evaluation Platform

To illustrate the difficulties of capturing all sources of

execution variability caused by interference, a brief description of the platform used in this paper, the Infineon AURIX Tricore [10] is given. The AURIX is a platform designed for use in real-time automotive systems, and has a number of features to facilitate reliable computation. However, as the AURIX is designed to host multiple tasks, with different requirements, the three processor cores of the AURIX each have different capabilities, suited for their intended roles. These capabilities are as follows:

- Core 0: Error checking, Energy Efficient Tricore 1.6E core
- Core 1: Error checking, High Performance Tricore 1.6P core
- Core 2: No error checking, High Performance Tricore 1.6P core

Each core has access to a crossbar which connects a 472KB SRAM unit, 4MB of flash memory, and any external peripherals. Further details about the processor cores, such as local cache or scratchpad configurations, are not publicly available, which complicates any analysis which requires this knowledge. Intercore interference is typically caused by contested accesses to one of the external resources; for example if two cores access the flash memory simultaneously they will contend as the flash cannot serve multiple requests simultaneously.

Each core also exposes its own 12 PMCs (9 in the case of Core 0), which have the capability to monitor performance metrics such as cache hits/misses, and pipeline stalls. PMCs configuration on different cores are independent, however each core only has 3 registers to monitor its PMCs. Furthermore, each individual PMC on a core can only be mapped onto a single register, and so not all combinations of 3 PMCs may be monitored simultaneously. For example, it is impossible to monitor the number of hits for both the instruction and data caches of Core 1 simultaneously.

In the best case, to capture data on all PMCs of the AURIX, it would be necessary to run each test four times, which is undesirable in that it increases the amount of testing that is required of the user. While it is possible to run experiments four times to gather all data for this platform, other platforms expose far more PMCs which makes gathering all PMC data infeasible (e.g. the P4080 platform [24] exposes approximately 128 PMCs, with 4 registers per core, and would require each experiment to be repeated 32 times). There is also a need to remove PMCs which do not contribute to the analysis, as the use of low-quality PMCs can result in poor quality models; for example, a computationally heavy task may not fetch much data, and therefore the number of data cache hits does not provide any useful information for analysis.

The PMCs exposed by the AURIX platform cannot monitor the number of accesses to the crossbar or each resource in isolation, e.g. shared variables accesses bypass the cache and thus neither hit nor miss. The maximum latency suffered by accesses to specific resources, e.g. as a result of arbitration between concurrent access, is also unclear. This limits the application of approaches such as [20] which rely on those two values to incorporate the contribution of interferences into WCET analysis.

Taking these points into account, it can be argued that it is desirable, and at times necessary, to reduce the number of PMCs to a smaller and more manageable set, and rely on existing obervable events to build an understanding of the impact of interferences. However, without prior knowledge of the usefulness of PMCs, it is necessary to build a small dataset with all PMCs in order to determine their usefulness. To this end, the next section details an automatic PMC selection phase, which uses PCA to find a set of PMCs which are capable of representing the variability of the data.

2.4 PMC Selection

Literature on statistical methods refers to techniques for reducing the number of dimensions, PMCs in our case, as either *dimensionality reduction* or *feature selection*. The goal for this step is to identify the PMCs which are correlated, and then select a set of representative PMCs which can be captured in a single trace while still describing the majority of the data. While it is inevitable that some detail in the data will be lost at this stage, the reduction in the amount of effort required to get a single data point enables more data to be collected, which in turn increases the amount of data used in the forecast model, and therefore the accuracy of the predictions made.

In order to accomplish this, the technique of *Principal Component Analysis* [12] is employed. PCA is a technique which identifies correlations within a dataset by finding the *Principal Components* of the data, with each PC describing the amount of variance attributed to each correlated vector. In the context of this work, an example PC is the number of accesses to a shared bus. Finding which PCs represent the most variance is normally most useful in reducing the complexity of the dataset. For example, if a PC accounts for less than 10% of the variance of the entire dataset, then this can be interpreted as sampling error and thus data along this PC can be ignored. The end goal of PMC selection is to find a small number of easily observable PMCs to measure. This small number is normally dictated by the number of PMCs available on the platform being used. An ILP solver [8] is used to determine the set of PMCs which represent the maximum amount of variance in the data set collected. A detailed evaluation of the PMC selection can be found in [14].

One issue that may be encountered as a consequence of PMC selection occurs if poor quality PMCs are selected; this can happen if user constraints prevent high quality PMCs from being selected, or high quality PMCs simply do not exist. That is, there are no significant effects from interference either due to restrictions on the contenders from the user or due to the design of the task itself. However, this does not lead to invalid results: If this is the case then the outcome of the algorithm is that model construction in the next stage of FBI will immediately fail, see section 2.5. The reason is there will be no link between interference and execution time. Hence one can simply conclude that for the task being evaluated, multicore interference does not impact the execution time; this can occur if, for example, all code and data for the task is consistently cached locally to the core, and therefore unaffected by multicore interference.

2.5 Modelling

Once enough data describing the major variations in the PMCs has been collected, i.e. the previous phase succeeds in selecting PMCs, it can be used to develop forecast models. In this paper, the technique used to construct the model is the TensorFlow Deep Neural Network implementation [6] via the Keras framework [1]; TensorFlow was chosen as it has a proven record on being

able to model complex data [6] and has well optimised implementations available. The desired output of these models is a multiplication factor which can be applied to a single-core execution time to give the corresponding execution time in a multi-core environment, subject to a given rate of interference from the contender cores. An important consideration for an analysis technique is that users of the approach are unlikely to appreciate having to repeatedly test the same path through a task under varying rates of interference to obtain an interference multiplier specific to that path. Further, this approach may be unsound, as even if this was carried out for the worst-case path of the task in a single core environment, there is no guarantee that the worst-case path of the task in a multicore environment is the same. Therefore, it is necessary to acknowledge that variability in the execution times of a task may come from the path taken and/or multicore interference, and that there must be a way to distinguish between the two.

FBI analysis does not record the path of the task under analysis, but does have information on the nature of the resources each path requires via the PMC data of the core under analysis. Hence, by converting the raw PMC data into a rate of change, corresponding to the rate at which the underlying resource is accessed, it is possible to compare the amount of resource each path requires. It is easy to surmise that if two paths differ in length but access resources at the same rate, then the effect of interference is likely to be proportionally the same. Conversely, if paths of the task access shared resources at different rates, then the effect of interference on these paths will be different.

Given that the rate of change of the PMCs' data for the core under analysis provides a useful measure of the requirements of the path of the task that was executed, it now follows to determine how the data from the PMCs impact the analysis. In the case that the data from PMCs does not correlate to the execution time of the path then this is simple: the resource being measured by the PMCs will not be selected by the PMC selection step. In the alternate case, where the data from the PMCs does correlate to the execution time then PMC selection will either select the PMC being monitored, providing direct information on the path taken to the analysis, or it will select a correlated PMC, providing indirect information on the path taken to the analysis. Therefore, in the case that the path taken changes the requirements or interactions of the task, FBI analysis is able to infer this information and process it accordingly. If the path does not have an effect on how the task responds to interference, then FBI analysis will ignore the information and focus on more relevant PMCs.

The next step is to find the baseline b for which interference multipliers are calculated; this is defined to be the shortest observed execution time, for which the interference multiplier is defined to be 1. The remaining data is sorted by the rate of change of each PMC's data into uniformly sized bins, and each bin is assigned an interference multiplier defined to be $\frac{a}{b}$, where a is the mean execution time of all measurements in the bin. The mapping of the rates of change of the PMC data to interference multipliers can then be used as the input for training the forecast model. A pseudocode implementation of this process is given in Algorithm 1. The value of *rate_block_size* depends on the range of the input values, and *min_bin_size* depends on random noise in the sample. In this paper, the values for *rate_block_size* and *min_bin_size* are chosen to be 0.01 and 10 respectively. These values were derived by trial and error, and have been shown to be appropriate across a range of tasks and platforms[2].

Due to the use of the shortest observed execution time as the baseline and the lack of detailed path information, the interference multiplier returned from the analysis may include the ratio between the shortest and longest paths. This can be corrected for when querying the model by limiting the PMCs monitored to those which are from interfering cores rather than the core under analysis[3].

```
1  Function GetAverageETs(data, rate_block_size,
     min_bin_size)
2      ets ← dictionary()
3      for execution_time, pmc_rates in data do
4          block ← [⌈pmc_rate/rate_block_size⌉ | pmc_rate ∈
             pmc_rates]
5          if block not in ets then
6              ets[block] ← []
7          append execution_time to ets[block]
8      Remove any entries in ets with less than min_bin_size
         execution times
9      output ← dictionary()
10     for each block in ets do
11         output[block] = mean of ets[block]
```

Algorithm 1: Algorithm for finding average execution times for given rates of interference as characterised by PMCs

(Benjamin: I do not know how much the renaming of most things to PMC helps with this one.)
It would have been helpful to provide a better context to its applicability in real-time systems. For example, reading the paper it is sometimes confusing to know what are the actual input data to the FBI model to train and query it. The inputs are referred as "observable features", "interference rates", "rates of change of the PMC data", "rates of change at which the observed features change" among others, which are somewhat confusing and vague. Are the inputs an array of numbers? only one number? It would have been welcomed to have an example in which these values were explicitly stated, to get a better understanding of how FBI is used in this concrete application in realtime systems.

In order to accomplish the actual forecasting, Deep Neural Networks [6, 1] are employed. As seen in Figure 2, Deep Neural Networks can be trained to learn a function using input/output data for that function. In this case, the input data to the function is provided as the interference rates (derived from the PMCs), and the output is the calculated interference multiplier. For this work, an n-input, 1-output 32-node dense rectified linear TensorFlow network for regression learning was used with Poisson Regression used for the objective function, where n is the number of selected PMCs. The n-inputs relate to the factors measured via the PMCs, the 1-output is the predicted interference multiplier, and 32 nodes equates to a dense network which maximises the connectivity between the neural networks layers. These parameters were chosen

[2]This includes all the benchmarks presented in this paper for the AURIX, P4080 and quad-core ARMv8 from a Raspberry PI3. It also includes industrial examples from DENSO AUTOMOTIVE Deutschland GmbH on the AURIX and Ikerlan on the P4080.

[3]Interference local to a specific core can come from a variety of shared resources, however cache-related preemption is the principal one.

as either the most appropriate for this type of regression problem or were selected after experimentation determined that further increases in complexity did not yield further increases in accuracy. This configuration also has the advantage that it places a lower penalty on overestimation than underestimation when learning, meaning that Deep Neural Networks of this form will consistently be sound rather than unsound. Rectified linear refers to the fact the inputs are normalised to lie within the range of 0 to 1 which is always advised when training neural networks. The Poisson objective function is assumed as it tends to be a good loss function when dealing with outputs that can have a large variation in scale. By contrast, other applicable loss functions, e.g. mean squared, tends to struggle with numbers spread over a large range [17].

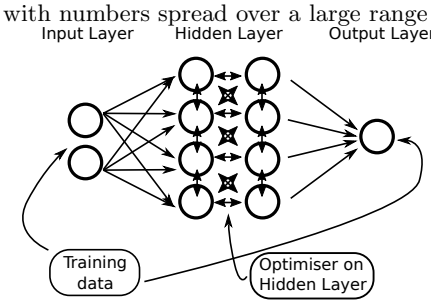

Figure 2: Use of Deep Neural Networks

The use of deep neural networks enables the automated learning of the relationship between interference measurements and their effect on the execution time of a task. One common pitfall in the application of machine learning is the risk of overspecialisation due to poor quality training data. This can be avoided by the use of *Ensemble Modelling* [2]. Ensemble Modelling is an intuitively simple technique: whenever a model is constructed, there is a random chance that the model is inaccurate for any given portion of the input space. Provided that this random chance is less than 50%[4], and multiple models are constructed in the same way and the consensus of the models is used to determine the result, then the chance of the model being inaccurate is decreased.

FBI analysis takes advantage of Ensemble Modelling by training multiple Deep Neural Networks from the data gathered. The training data is first split into equally sized blocks with which to train the deep neural nets, using distinct training data. As the training of the Deep Neural Networks is handled by the Keras library [1], it is not detailed in this paper. The use of multiple sets of training data allow the Ensemble Model to be populated by multiple distinct models to minimise the risk of overspecialisation. As this paper is concerned with learning a normalised multiplier for execution times, the consensus of the ensemble model is calculated using the Geometric Mean [25]. The Geometric Mean is used as each model may be normalised to a different value, which implies other means such as the Arithmetic Mean are not valid.

2.6 Trusting the Model

As the model is constructed by machine learning, and is thus not easily understandable, the model cannot be trusted in the same way that traditional analysis techniques can be trusted[5]. To achieve a level of trust in the model, the idea of Forecast Accuracy is used [16]. Forecast accuracy is a well established metric that frames the accuracy of predictions as an easily determined test; the model is tested by comparing predictions to observations which were not used to construct the model. Depending on how accurate the model is when compared to reality, the forecast accuracy can be established. Further, the statistical confidence required of the forecast accuracy can be set arbitrarily, depending on the amount of data available.

In this application, evaluating the forecast accuracy amounts to comparing results from actual observations with the results from the FBI model. Assuming that there is no systemic bias in the model, the error can be assumed to follow a normal distribution around the true value. The degree of confidence required by the user thus bounds the acceptable observed absolute error in the model, defined as the margin for error E. If the FBI model is sufficiently accurate for all queries, with the degree of accuracy corresponding to an acceptable margin of error E specified by the user, then a claim can be presented that the model is accurate to a degree of accuracy given by the number of tests conducted. For example, if FBI analysis determines its forecast accuracy at the 10% level is 99%, a margin for error of 10% will hold for 99% of the data; 99% of the model predictions are within $\pm 10\%$ of the actual value. As this is a simple operation, forecast accuracy is completely automated and calculated after the generation of the model, allowing practitioners a degree of confidence in the results. Note the FBI method does not aim to obtain an absolute WCET bound, but rather a WCET estimate that is valid with a reasonable degree of statistical confidence.

2.7 Integrating the Components of the Analysis

Having defined the individual components of FBI analysis, it remains to outline how these components are combined. Firstly, a small sample of heavily instrumented data is provided to the PMC selection component (Section 2.4). Once appropriate PMCs are selected, the bulk of the data is collected; provided that instrumentation is automated, there is no need for manual intervention at this step. A portion of this data is used to generate an initial model (Section 2.5). This model is then tested with other observations (Section 2.6) to determine the confidence we have in the model, and therefore the appropriateness of using the model. This process is outlined in Algorithm 2.

In order to make a prediction using an FBI model, each Deep Neural Net in the Ensemble is queried with the given interference rates. The results of each Deep Neural Network are combined into an Ensemble Average by taking their Geometric Mean. This Ensemble Average is then returned as the predicted interference multiplier for the requested interference rate. However, even once all Deep Neural Nets are constructed, it is necessary to find a margin for error for the FBI model. This is accomplished by testing the model with previously unseen observations and computing the resulting error. Once this is computed, the margin for error should be added to any interference multiplier derived from the FBI model, which in turn allows us to have confidence in the accuracy of the results. Next, the FBI model can be used to compute the

[4]If this is not true, then the Ensemble becomes more inaccurate than its constituent parts, resulting in an easily detectable failure of the technique.

[5]Although it should be noted that no analysis technique can be deemed completely trustworthy, due to the potential for implementation error.

```
Function FindConfidence(model, dataset, target_accuracy)
    accurate ← all results in dataset such that
      abs((model.predict(results.factors)/observed_interference)−
      1) < target_accuracy
    return len(accurate)/len(dataset)
Function FBITrain(simultaneous_pmcs, bin_size,
  no_of_nets, test_samples, target_confidence)
    initial_dataset ← data captured with all PMCs
      instrumented
    best_pmcs ← GetBestPMCs(initial_dataset,
      simultaneous_pmcs)
    main_raw_dataset ← data captured with best_pmcs
      instrumented
    main_dataset ← GetAverageETs(main_raw_dataset,
      bin_size)
    partition main_dataset into test_dataset of size
      test_samples and train_dataset
      model ← EnsembleModel(, no_of_nets, train_dataset)
    Find minimum margin_for_error such that
      FindConfidence(model, dataset, margin_for_error) >
      target_confidence
    return model, margin_for_error
```

Algorithm 2: The FBI Training Algorithm

interference multiplier for a given set of interference rates. The interference multiplier, I, and margin for error, E, can then be applied to the single-core WCET estimate in order to find a WCET estimate with given interference, as in the following equation:

$$WCET_{inteference} = WCET_{single\_core} \times (E + I) \quad (1)$$

In order to find the WCET with estimate interference, it is necessary to find the worst case interference multiplier (WCIM). As the configuration of interference rates needed for the WCIM may not be the maximum of each interfering source, the simplest way to do so is to search over the FBI model. Such a WCIM is likely to be an over approximation due to configurations of interference rates which do not appear in the deployed system. However, such a WCIM is useful in parallel development, as it allows an approximation of the WCET with interference even when the contender tasks are unknown. Once contender tasks are available, the WCIM can be refined based on the rates of interference those tasks actually produce, and FBI still shows a benefit as it is not necessary to repeat the tests on task under analysis; instead one can search over the FBI model, but this time constrained to the levels of interference observed from contender tasks. Even when the contender tasks are available, FBI provides useful information since as the system is integrated it becomes harder to test for all the possible cases that may be seen post-deployment. In contrast, FBI provides a convenient means for understanding how the system behaves given bounded ranges of shared resource use.

3. EVALUATION

To test the FBI approach, various benchmarks were deployed on the AURIX platform. Instrumentation was provided by using the Rapita Verification Suite [22] to insert customised instrumentation points before and after the execution of a task, which allow the PMCs of the AURIX to be read. These were then processed to reveal the rate at which the PMCs changed, providing the input to the method. For simplicity, this work assumes non-preemptive tasks as this simplifies the experimental setup. The FBI method does not impose any constraints that prohibit preemptive tasks. Benchmarks from the Taclebench suite [3] were investigated. As the AURIX platform has two distinct types of core, each benchmark was run on both the energy efficient Core 0 (labelled as C0) and the high performance Core 1 (labelled as C1).

Interference was provided by synthetic contenders running on all cores. The synthetic contenders were implemented by accessing uncached memory addresses at a randomly selected frequency to provide a range of interference values. Care was taken to ensure repeatability between different configurations of contenders (Section 2.2).

In each test, full instrumentation of all PMCs was supplied for 1000 samples, which was used to determine which PMCs should be gathered, for varying numbers of PMCs and whether or not data gathering should be restricted to a single run of the task. The sample size was set to 1000 as this was observed to consistently return the same results from PMC selection as larger samples, indicating the point of diminishing returns. Once the relevant PMCs were determined, each FBI model was trained using 6000 samples, with Ensemble models being composed of 3 sub-models formed by Keras models of the form given in Section 2.5; 6000 samples was found to be the minimum number required to achieve consistent results. The collection of all required 8000 samples (1000 for PMC selection, 5000 to complete the FBI model training dataset, and 1000 to evaluate the forecast accuracy) relies on an automated process which took less than 30 minutes per test. This amounts to a minimum of $(4 \times 1000) + (5000 + 1000) = 10000$ runs of the target task. The number of required runs to build a complete dataset depends on the available PMCs (P) and registers (R) on the platform and the number $P' < P$ of selected PMCs: $\lceil (P/R) \rceil * X + \lceil (P'/R) \rceil * X'$, where X captures is the number of samples required for PMC selection (§ 2.4), and X' the samples used to complete the dataset for training ($5000 + 1000$ on the AURIX).

The resulting models were then evaluated against 1000 unseen observations, which enables sufficient statistical confidence, to determine the following properties:

1. *Absolute Error*: The distribution of the absolute errors seen, including the minimum and maximum errors. These results are then used to find an appropriate margin for error, defined such that the margin for error is the lowest value that can be added to predictions made by the model which guarantees that the results upper bound the unseen observations, i.e. the execution times for test cases yet to be collected, with a given statistical confidence.

2. *Execution Time Distribution*: The execution time distributions of No Interference (task run in isolation), Observed Interference (task run with given interference rates) and Max Interference (task run with maximum possible interference), were compared to the results from the FBI approach to determine the accuracy of the method.

In reporting the results, individual results are referred to by the following scheme: *benchmark name-core executed on-no. of PMCs selected in analysis restricted to the number gathered in a Single Run (SR) or Multiple Runs (MR).*

Figure 3 gives an overview of the forecast accuracy across all selected benchmarks for varying configurations of analysed core, number of analysed PMCs, and PMCs gathered across a single or multiple runs. The main contributors to execution time variability depend on the analysed task, PMC selection is thus performed for each

task and configuration[6]. These results demonstrate that when setting a margin for error of 10% the interference multipliers are accurate, with a statistical confidence of 99.99%; the error can be reduced or the confidence improved by increasing the number of tests conducted for all configurations. However, for all benchmarks apart from *matmult*, this margin can be decreased to 1% at the same confidence, i.e. 99.99% of predicted values are within 1% of the actual value. *matmult* exhibits interesting behaviour in that the benchmark is more predictable on C1 than on C0. There are two reasons for this: the additional optimisations of the higher performance C1 yielding more predictable behaviour; and the slower C0 not being able to generate as much interferences.

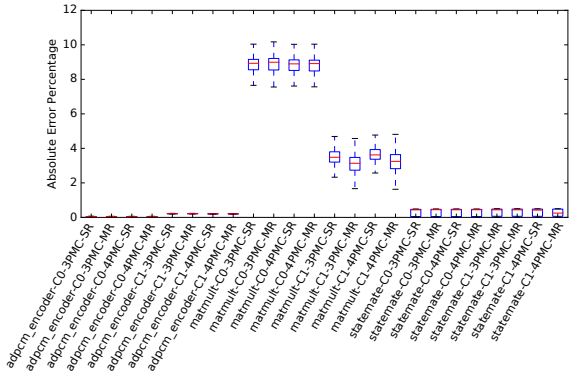

Figure 3: Error Distribution for Simple Benchmarks

Experiments have been performed using traditional examples of loads. The results seem to achieve good results. However, the figure 4 a) shows a significant dispersion between actual, prediction and prediction + margin.

FBI can both overestimate and underestimate, therefore applying always the margin of error can be counterproductive in some cases. How often does FBI without the margin of error overestimate?

Figure 4 focuses on the "worst behaving benchmark matmult on C1". But looking at the results of Figure 3, isn't the worst behaving one "matmult on C0"?

We focus on the worst behaving benchmark *matmult* on C1 using 4 PMCs. The PMCs, selected during the PMC selection phase (§ 2.4), capture cache hits and memory stalls suffered by the contending Core 0 and Core 2. The Figures 4 and 5 give an overview of FBI analysis performance. Figure 4 presents the direct comparison of actual and predicted interference multipliers across different experiment runs of *matmult*, i.e. under randomised contenders and thus interferences. While results vary in amplitude, it shows that FBI analysis is capable of predicting the peaks and troughs of interference, e.g. between runs 5 to 10. Further investigation revealed that FBI analysis tends to over-approximate this benchmark. If only the optimistic analysis results (i.e. those where FBI predicts an interference multiplier less than the observed one) are considered, then the absolute error distribution falls to 6%.

Applying the FBI technique to each execution time collected without interference, as per Equation (1), yielded the distribution shown in Figure 5. Figure 5 captures the likelyhood that an run of the benchmark with fixed, randomised or predicted interferences, exceeds a specific execution time. For example, more than 80% of the runs under variable interference exceed 400 KCycles. Under a fixed interference configuration, e.g. No interference or Max interference, all runs adhere to a single bound. As can be seen, when used to transform single-core execution times (No interference) to multicore with interference execution times (FBI) , FBI adds a small amount of overhead due to its margin for error and hence consistently upper bounds the actual observations.

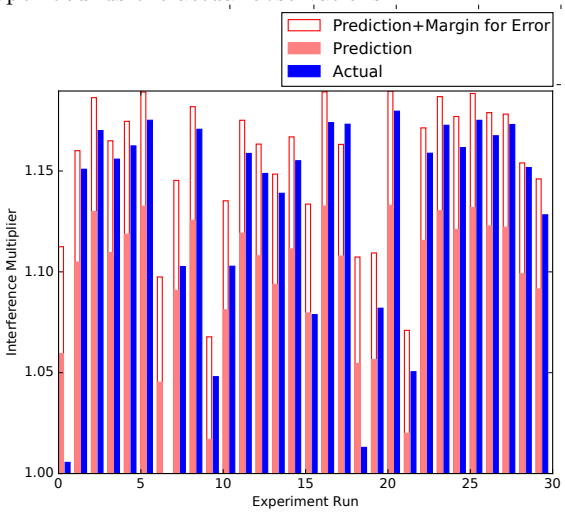

Figure 4: Comparison of Prediction and Actual results with a margin for error of 6%

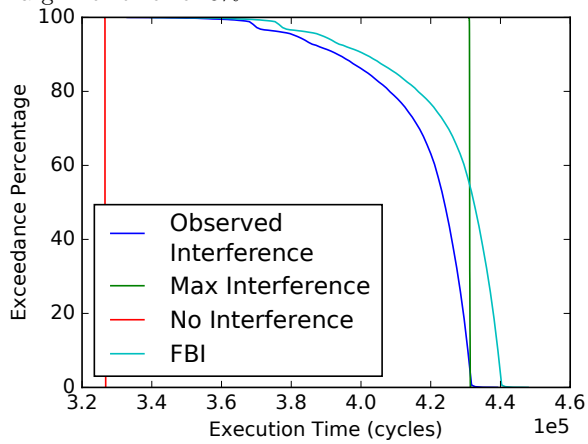

Figure 5: Execution times calculated by the FBI method

4. CONCLUSION

This paper presents an initial attempt to apply deep machine learning algorithms to the multi-core interference problem by hypothesising a link between the PMCs available on a processor and the rate of interference a task suffers. As the exact nature of this link varies between processors and applications, machine learning was used to determine the nature of this link in a specific context. The evaluation demonstrated that this approach is applicable to the Infineon AURIX processor family, and capable of producing a model of interference multipliers along with

[6]PMC selection has been discussed in [14] and is omitted for the sake of brevity.

the statistical confidence in their correctness. This model can then be used to derive a maximum execution time including interferences to a given statistical confidence.

Acknowledgments

This work was partially funded by EU FP7 IP PROXIMA (611085) as well as the UK EPSRC Projects MCCps (EP/P003664/1) and STRATA (EP/N023641/1). EPSRC Research Data Management: No new primary data was created during this study.

5. REFERENCES

[1] F. Chollet. Keras, 2015.

[2] M. Collins. Ensembles and probabilities: a new era in the prediction of climate change. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 365(1857):1957–1970, 2007.

[3] Contributors. Taclebench benchmark suite. http://www.tacle.eu/index.php/activities/taclebench.

[4] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Symposium on Principles of Programming Languages*, 1977.

[5] J. M. Anderson et al. Continuous profiling: Where have all the cycles gone? *ACM Trans. Comput. Syst.*, 15(4):357–390, November 1997.

[6] M. Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[7] D. Griffin, B. Lesage, F. Soboczenski, I. Bate, and R. I. Davis. Modelling fault dependencies when execution time budgets are exceeded. In *Proceedings of the 23rd International Conference on Real-Time Networks and Systems*, 2015.

[8] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2015.

[9] MIPS Technologies Inc. R10000 microprocessor user's manual-version 1.1. http://techpubs.sgi.com/library/manuals/2000/007-2490-001/pdf/007-2490-001.pdf.

[10] Infineon. *Aurix (TM) Family TC27xT Documentation*, 2014.

[11] W. Jing. Performance isolation for mixed criticality real-time system on multicore with xen hypervisor. Master's thesis, Uppsala University, Department of Information Technology, 2013.

[12] I. Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.

[13] S. Law and I. Bate. Achieving appropriate test coverage for reliable measurement-based timing analysis. In *28th Euromicro Conference on Real-Time Systems*, pages 189–199, 2016.

[14] B. Lesage, D. Griffin, I. Bate, and F. Soboczenski. Exploring and understanding multicore interference from observable factors. In *International Conference on Automotive - Safety & Security*, To Appear.

[15] Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean. A statistical response-time analysis of real-time embedded systems. In *Proceedings of the 33rd Real-Time Systems Symposium*, 2012.

[16] D. C. Montgomery, L. A. Johnson, and J. S. Gardiner. *Forecasting and time series analysis*. McGraw-Hill Companies, 1990.

[17] M. A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[18] F. Farshchi P. K. Valsan, H. Yun. Taming non-blocking caches to improve isolation in multicore real-time systems. In *Real-Time and Embedded Technology and Applications Symposium*, 2016.

[19] M. Paolieri, E. Quiñones, F. J. Cazorla, G. Bernat, and M. Valero. Hardware support for WCET analysis of hard real-time multicore systems. *SIGARCH Comput. Archit. News*, 37(3):57–68, June 2009.

[20] M. Paolieri, E. Quiñones, F. J. Cazorla, R. I. Davis, and M. Valero. IA^3: An interference aware allocation algorithm for multicore hard real-time systems. In *17th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 280–290, 2011.

[21] P. Radojković, S. Girbal, A. Grasset, E. Quiñones, S. Yehia, and F. J. Cazorla. On the evaluation of the impact of shared resources in multithreaded COTS processors in time-critical environments. *ACM Trans. Archit. Code Optim.*, 8(4):34:1–34:25, 2012.

[22] Rapita Systems. Rapita verification suite. https://www.rapitasystems.com/.

[23] D. Sanchez and C. Kozyrakis. Vantage: scalable and efficient fine-grain cache partitioning. In *ACM SIGARCH Computer Architecture News*, volume 39, pages 57–68. ACM, 2011.

[24] Freescale Semiconductor. Eref: A programmer's reference manual for freescale embedded processors.

[25] L. J. Stephens. *Schaum's Outlines: Beginning Statistics*. McGraw-Hill, 2nd edition, 2006.

[26] H. Yun, R. Pellizzon, and P. K. Valsan. Parallelism-aware memory interference delay analysis for cots multicore systems. In *27th Euromicro Conference on Real-Time Systems*, 2015.

[27] X. Zheng, L. John, and A. Gerstlauer. Accurate phase-level cross-platform power and performance estimation. In *ACM/IEEE Design Automation Conference*, 2016.