
# Timing Analysis and Verification of Multi-core Real-Time Systems for Aerospace Application

**Iain Bate**

**Dept of Computer Science**

**University of York**

**iain.bate@york.ac.uk**

# Overview of the Seminar

- **Current driving forces for industry**
  - Mixed-criticality systems to reduce development costs
  - Multi-core to enhance functionality and for supply reasons
  - Requirements from CAST-32A
- **What are we currently undertaking**
  - Assessment of processor issues
  - Multi-core analysis
  - Task scheduling and allocation
- **Future directions**

# Industry's Driving Forces

- **Mixed Criticality Scheduling allows low criticality tasks to execute on the same target as high criticality tasks**
  - Allowing low criticality tasks to have deadlines, periods and other timing requirements
  - Giving a good balance between safety, flexibility and maximising utilisation
- **For example, Rolls-Royce have a number of tasks considered high criticality however can be disabled for short periods of time**
  - For instance recording error logs in non-volatile memory, a time consuming but still important process
- **Low-criticality software may be developed to the same integrity but not verified to the same degree**
- **Principal benefits – Cost & Flexibility**

# Industry's Driving Forces

- **Multi-core processors are widely used in automotive systems**
  - Feeling is they are inevitable for avionics
- **Broad spectrum of anticipated uses and types of processor considered**
- **Motivations range from physical resources through need for more compute power to supply availability**
- **Seen as one of the biggest challenges as to date arguably no mature (accepted) academic approach**

# Industry's Driving Forces

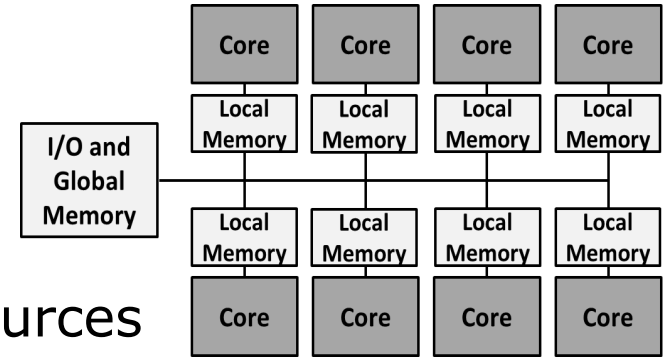
- **There are many common myths around certification**
  - E.g. static WCET analysis is needed for DAL-A
- **Certification standards and guidance rarely specifically mentions timing**
- **Certification standards do talk about**
  - Understanding and mitigating risks
  - Establishing *validated* requirements and verifying them
- **Software standards do not discuss likelihoods of failure**
  - We should meet requirements but the following may be acceptable
  - 99% of the time task X completes at a rate of Y but skipping a job doesn't cause Z
- **Challenge is we don't know what is acceptable**

# Industry's Driving Forces

- **Certification authorities are giving guidance which would be hard to literally achieve**
- **CAST-32A states** *MCP_Resource_Usage_4: The applicant has identified the available resources of the MCP and of its interconnect in the intended final configuration, has allocated the resources of the MCP to the software applications hosted on the MCP and has verified that the demands for the resources of the MCP and of the interconnect do not exceed the available resources when all the hosted software is executing on the target processor*
- **This moves us away from Black Box analysis**
- **Supports our usual approaches**
  - Design-for-predictability
  - Design-for-safety

# Why are multi-cores so difficult?

- **Typically designed for good average case performance at low cost and so hardware resources are shared between cores**
  - Interconnect / bus
  - Caches (multiple levels)
  - Main memory (e.g. DRAM)
  - and various other HW specific resources




- **Software components running on different cores contend for shared HW resources**

# Why are multi-cores so difficult?

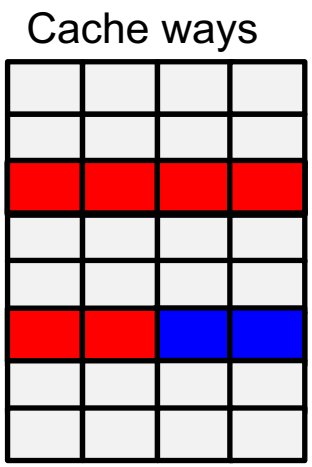
- **The interference on execution times resulting from this contention can be very difficult (if not impossible) to predict**
- **Even within a family of devices, e.g. AURIX, the variants have subtle but important differences**
- **The choice of platform (processor family, variant and RTOS / bare board) and how it is configured is fundamental**
- **Chicken and egg situation as choices depend on software**
  - Software hasn't been written yet
  - The way the software is written influenced by choices

# Improving predictability

- **Choosing multi-core hardware with predictable arbitration policies for access to shared hardware resources**
- **Choosing configurations that favour predictable performance, for example by dividing (i.e. spatially or temporally partitioning) shared resources**
- **Using course grained forms of time partitioning to control when different software components access shared hardware resources (e.g. via code refactoring and/or scheduling policies)**
- **Regulation: monitoring and constraining the number of resource accesses from each core in a given time period or in a given time partition**

# Improving predictability

- **Caches and scratchpads**
- **Caches in multi-core systems**
  - Many issues are common with the use of caches in single-core systems – focus only on inter-core effects due to shared caches
- **Use of cache partitioning**
  - Essential to partition shared cache to prevent SW components on different cores evicting each others  and avoid cache thrashing

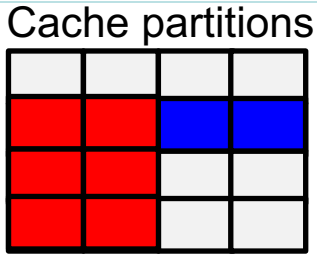


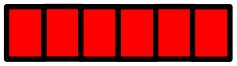

Cache thrashing – unpredictable, induces many memory accesses, very low performance

Cache partitioning – increases WCET w.r.t lone use of whole cache but much more predictable and can improve performance over shared cache with interference

# Improving predictability

**Cache partitioning**



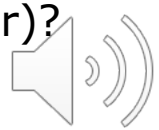

**A cautionary tail…**

- Execution on a core with a dedicated cache partition can suffer over **20x** slowdown due to co-runners accessing other cache partitions
- Why? and what caused this unexpected behaviour?
  - Some architectures can process instructions out-of-order and have non-blocking caches - behaviour is managed via cache Miss Status Holding Registers (MSHRs)
  - MSHRs are a (hidden) shared resource - if all MSHRs are in use the accesses will stall even if they are cache hits in a different partition!
- Solution is to also partition the MSHRs between the cores

**Important**: need to investigate all possible interference channels and verify that no unexpected interference occurs

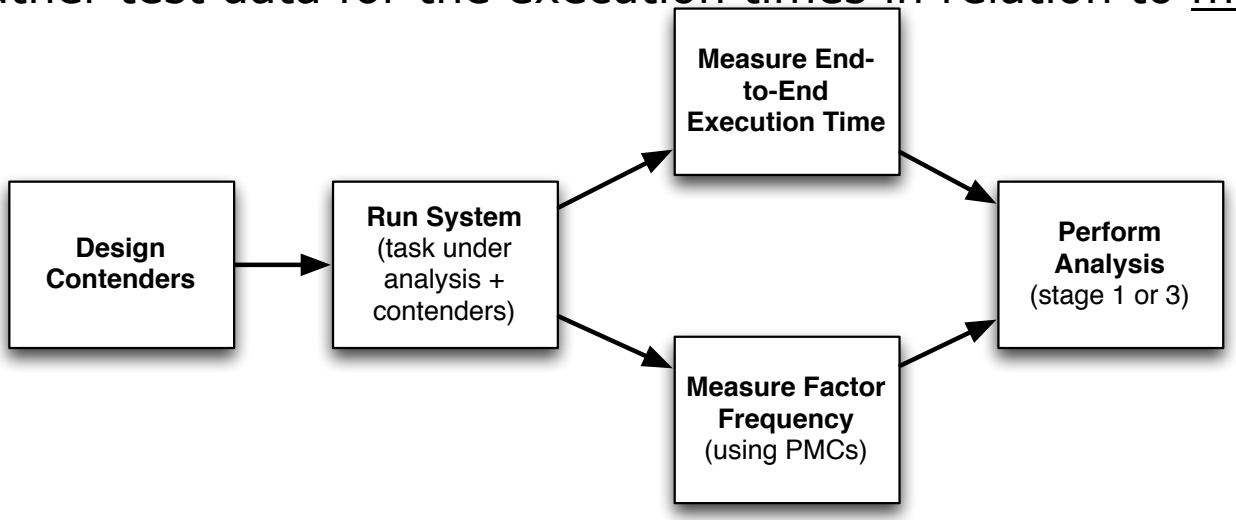
# Guidance on selection of hardware

- **Gain in-depth understanding of platform, including detailed behaviour of all its shared hardware resources**
- **Identify and understand all the potential interference channels due to shared hardware resources**
- **Choose hardware configuration and employ software techniques and mechanisms to improve predictability to mitigate interference**
- **For each interference channel verify the impact of configuration options, resource management policies, and mitigation mechanisms**
- **Appraisal and selection**
  - Can full isolation be achieved? (unlikely in practice)
  - Does partial isolation bound maximum interference independent of co-runner behaviour?
  - Is the interference unconstrained (dependent on co-runner behaviour)?
  - Verify slowdown in performance and absolute performance achieved



# Multi-Core Timing Analysis

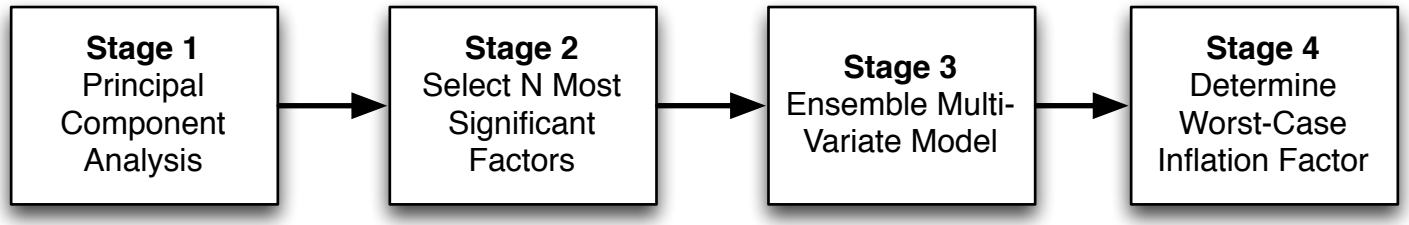
- **Steps 1&2 – Deciding most significant factors**
  - Gather test data for the execution times in relation to most factors




  - Step 1. Use Principal Component Analysis to determine the significant factors
    - Significant - those causing greatest variability in the execution time
  - Step 2. Select number of significant factors dependent on:
    - Number of performance counters
    - Most strongly correlated with variations in the execution time

# Multi-Core Timing Analysis

- **Steps 3&4 – Build and Query multi-variate model**
  - Perform much more testing for the significant factors
  - Ensemble from a range of machine learned models
  - Determine the accuracy of the model by comparing observations and prediction
  - Use accuracy to direct further testing
  - Step 4: Extract worst-case inflation factor (HWM) by search over model




- **Analysis stages**

# Multi-Core Timing Analysis

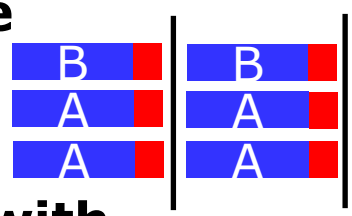
- **Currently have a rig of 20 Raspberry PI3Bs**
- **Using Rolls-Royce FADEC software**
- **Aim is to**
  - Better understand sources of interference
  - Further develop machine learning for multi-core
  - Develop evidence of accuracy and sufficiency of analysis results
  - Make an argument suitable in the context of CAST-32A
- **The key to acceptance may be integration of**
  - Systematic testing including contenders
  - Statistical analysis of prediction accuracy

# Types of Interference

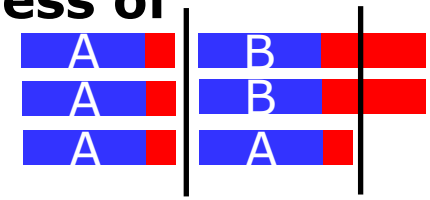
- **No dependence: Ideal but rare / unrealistic for shared hardware resources on COTS multicores**
  - No additional interference as such BUT standalone execution time typically made worse by policies used to ensure no dependence (effectively the interference is always there irrespective of what the co-runners do – since the resource is **divided**)
- **Direct interference: Takes up some of the bandwidth of the resource – the resource is shared**
  - Additive effect – interference equates to co-runners time using the resource
- **Indirect interference: Alters the state of the resource adding to the operations required**
  - Super-additive effect – interference equates to co-runners standalone time using the resource PLUS additional time for extra or slower operations by both the task of interest and its co-runners
- **Hidden interference: Alters the state of the resource in such a way that stalls or causes unacceptably long / unbounded delays to the task of interest**
  - This type of interference needs to be identified and eliminated

# Nuanced task model for multi-core

- **Enables task allocation, schedule construction, and analysis to give predictable and efficient use of hardware**
- **Parameters considered as budgets that individual tasks, or a group of tasks executing within a time partition supporting composition which enables independent development by different teams**




- **Budgets support development of robust designs with headroom for modifications**
- **Model can also be used to study the effectiveness of different hardware configurations (design optimisation)**




- **IMPORTANT to have nuanced context-aware information that enables engineers to explore and compose robust designs**

# Planned Work

- **Hi-Class is a large project with most of UK civil avionics**
- **Key driver is multi-core for avionics**
  - Low numbers of predictable cores
  - Bare metal
  - 653 and non-653 based RTOSs
- **UoY providing advice on**
  - What information is needed from multi-core timing analysis
  - Testing strategy to gain this information
  - Architectural options for multi-core
  - Appropriate scheduling approaches
- **UoY is mainly investigating task allocation and scheduling of multi-core systems**

# Planned Work

- **Predictability is important here however maintainability is going to be fundamental**
- **Need to harness certification principles of**
  - As Low As Reasonably Practicable (ALARP)
  - ALARP – basically the cost of improving safety further / gathering more evidence shouldn't be disproportionate to the cost
  - Globalement Au Moins Aussi Bon (GAMAB)
  - GAMAB - basically no less safe than before
  - No less safe than before is an often used tactic but this needs translating to multi-core

# Planned Work

- **MOCHA is a Huawei funded project**
- **Much more complex software and platforms**
- **Number of similarities**
  - Jitter is important
  - Software has transactions
  - Overheads can’t be ignored
  - Different tasks have different importance
  - Static WCET analysis not practical
- **Again, control provided by scheduling and allocation with awareness of execution behaviour**

# Planned Work

- **Key research challenges across all work**
  - How to systematic test and understand sufficiency?
  - How to assess confidence in analysis?
  - What should the real requirements be?
  - How to create a valid digital twin to understand typical timing behaviours and quality of service?
  - What the argument and evidence should look like for certification?

# Acknowledgements


- **Members of the RTS group**
- **Rolls-Royce for years of support, inspiration and access to real systems**
- **Innovate UK for funding the Hi-Class project**
- **Innovate UK for funding the ATICS project**
- **Huawei for funding the MOCHA project**
- **Members of WashU for stimulating discussions and space to think**