Towards Certifiable Resource Sharing in Safety-Critical Multi-Core Real-Time Systems

Benny Akesson
Czech Technical University in Prague

Jan Nowotsch
Airbus Group Innovations
Embedded systems get increasingly complex
- Increasingly complex applications (more functionality)
- Growing number of applications integrated in a device
- More applications execute concurrently
- Requires increased system performance without increasing power

The resulting complex platforms
- are (heterogeneous) multi-core systems to improve performance/power ratio
- Resources in the system are shared to reduce cost
Some applications have **real-time requirements**
- WCET must be smaller than deadline

Applications have different **design assurance levels** (DAL)
- DAL level determines required certification effort [1,2]
- High DAL levels are very expensive and time-consuming to certify

Commercial-of-the-shelf (COTS) platforms are used
- Custom hardware not cost-effective with low volumes

Increased integration implies **mixed-criticality systems**
- Applications with different DALs share resources

Resource sharing creates **interference** between applications
- Makes it difficult to derive WCET of applications
- Highest DAL of applications must be used unless there is **isolation** [1]

Both **temporal and spatial isolation** is required [1]
- Applications must be "sufficiently" independent

Isolation on single core is typically provided by operating system
  - E.g. based on ARINC-653 specification [3]
  - "Robust" partitions created for sets of applications

Temporal isolation using time-division multiplexing (TDM)
  - TDM non-work-conserving (nwc) to eliminate interference
  - Application-level scheduling within a partition

How to ensure that applications sharing resources are isolated and that WCET of applications can be computed in certifiable mixed-criticality multi-core systems?

This presentation discusses this problem in a survey-like manner.
Presentation Outline

Introduction

Time-Predictable Hardware

COTS Analysis Methods

Airbus isWCET Approach

Conclusions
CompSoC is a platform for real-time applications [4]
- For independent app. development, verification, and execution

Components of tiled architecture [5]
- Processor tiles with MicroBlaze cores
- Æthereal network-on-chip
- Memory tiles with SRAM or SDRAM
- Peripheral tiles

Platform implementation in VHDL [6]

— **All resources are shared** [7]
  - NWC TDM partition scheduling on CPU (ARINC-653)
  - NWC pipelined TDM flit scheduling in network-on-chip
  - NWC TDM trans. scheduling or any scheduler + delay for DRAM

— **Performance analysis**
  - Data-flow models for all software/hardware components
  - WCET for all tasks/transactions

Extremely robust partitioning [7]
- Not a single cycle interference from other partitions
- Similar to PREcision-Timed Architectures (PRET) [8]

It is possible to design time-predictable multi-core platforms
- Extremely robust partitioning
- WCET for all tasks/transactions, but
- Average-case performance suffer

Application domain is practically restricted to COTS platforms
- Hardware is given
- Transfering technology is very difficult
- Most customers are oriented towards average-case performance
Introduction

Time-Predictable Hardware

COTS Analysis Methods

Airbus isWCET Approach

Conclusions
Analytically modeling a COTS platform is very difficult
- Hardware is optimized for average-case performance
- No detailed documentation of implementation
- Limited possibilities for measurements during validation
- Difficult to guarantee correctness / conservativeness of model

Often pessimistic assumptions about memory controller:
- Unknown size of reorder buffer in memory controller [9]
- Unknown work-conserving memory scheduler [10,11,12]
- Bounds still useful?

There is much work on bounding interference between tasks
- Vary w.r.t. task model and (task/transaction) schedulers

Common assumptions
- **Single outstanding transaction**
- No or partitioned caches
- **Different path of worst-case memory accesses** (WMA)

Abstraction of memory accesses
- Number of memory accesses per task / block [11,13]
- Minimum / maximum requests in interval [12] (for self / others)

Throttling popular to control memory interference [11,14,15,16]
- Can be implemented at operating system level
- Relies on good performance monitoring counters

Basic idea:
1. Assign memory access budgets
2. Monitor number of memory accesses using performance counters
3. Enforce budget by suspending tasks with depleted budgets

Optionally, there are mechanisms for slack distribution
- Observed slack [15] or proven slack [16]

→ New scheduling theory on top of memory throttling [13,17]
  – Respecting both memory budget and CPU scheduling

→ Theory requires knowledge about memory access times
  – Commonly done by assumption
  – Sometimes by measurements on platform [11,13]
  – Never done using validated analytical model

Measurement-based approaches offer pragmatic solution

Possible to use measurement-based WCET tools
- E.g. RapiTime
- Measure your way around things you cannot model

Stressing shared resources
- Possible using synthetic resource stressing tasks [18,19]

Introduction
Time-Predictable Hardware
COTS Analysis Methods
Airbus isWCET Approach
Conclusions
Setup
- Freescale P4080 multi-core platform
- SYSGO Pike OS operating system
- AbsInt aiT static analysis tool
- EEMBC Automotive benchmarks

- Individual core-local and interference analyses
- Separation of timing and resource analyses

Core-local Analysis

- Application Binary
- Control Flow Construction
- Control Flow Graph
- Control Flow Analysis
- Annotated Control Flow Graph
- Micro-Architecture Analysis
- Path Analysis

Process   Data
Core-local Analysis
Core-local Analysis

Interference Analysis

...
Comparison to intuitive approaches (minimum $t_{\text{min}}$ and maximum $t_{\text{max}}$ contention)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>$T_{\text{min}}$[ms]</th>
<th>$T_{\text{max}}$[ms]</th>
<th>$T_{\text{is}}$[ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>cacheb</td>
<td>114</td>
<td>1996</td>
<td>493</td>
</tr>
<tr>
<td>iirflt</td>
<td>60</td>
<td>136</td>
<td>116</td>
</tr>
<tr>
<td>rspeed</td>
<td>233</td>
<td>4468</td>
<td>612</td>
</tr>
<tr>
<td>a2time</td>
<td>29</td>
<td>524</td>
<td>231</td>
</tr>
<tr>
<td>bitmnp</td>
<td>154</td>
<td>262</td>
<td>225</td>
</tr>
<tr>
<td>tblook</td>
<td>122</td>
<td>449</td>
<td>289</td>
</tr>
<tr>
<td>matrix</td>
<td>21</td>
<td>35</td>
<td>32</td>
</tr>
<tr>
<td>aifftr</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>
Dynamic adaptation of resource budgets based on actual system progress

Progress determined through monitoring

Increased integration drives transition to multi-core platforms
- Resource sharing causes *interference* between applications
- Nightmare w.r.t. certification
- Problem to isolate sharing applications and safely determine WCET

Time-Predictable Platforms have been demonstrated
- Extremely robust partitioning and easy to determine WCET
- Difficult to get commercial uptake of technology

Analysis of COTS systems active research topic
- Difficult to model analytically due to lacking openness
- Community is finding the right models/abstractions
- Most models remain unvalidated
- Alternative is to use measurement-based techniques