Porting a safety-critical industrial application on a mixed-criticality enabled real-time operating system

Antonio Paolillo

Paul Rodriguez, Vladimir Svoboda, Olivier Desenfans, Joël Goossens, Ben Rodriguez, Sylvain Girbal, Madeleine Faugère, Philippe Bonnot

5th December 2017

5th International Workshop on Mixed Criticality Systems (WMC 2017)

Joint work
HIPPEROS and Thales R&T

Mixed-criticality?

The IMICRASAR project

Isolated MIxed CRiticality
Avionics System ARchitecture

Goals:

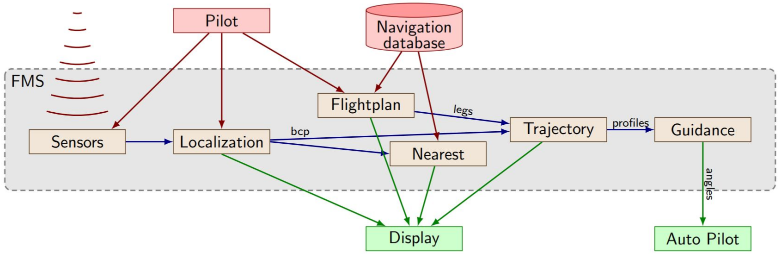
- create an isolated
  mixed-criticality
  hard real-time platform

  platform = OS + hardware

- support of an industrial application on the platform

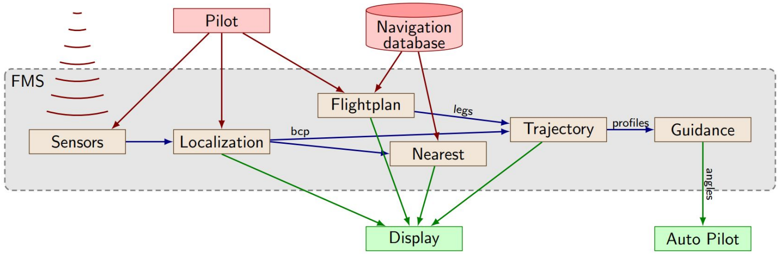
- results: retrieve unused CPU resource

In short, Thales brings us two components...



In short, Thales brings us two components…

and asks us to make a system out of it



Possible solutions

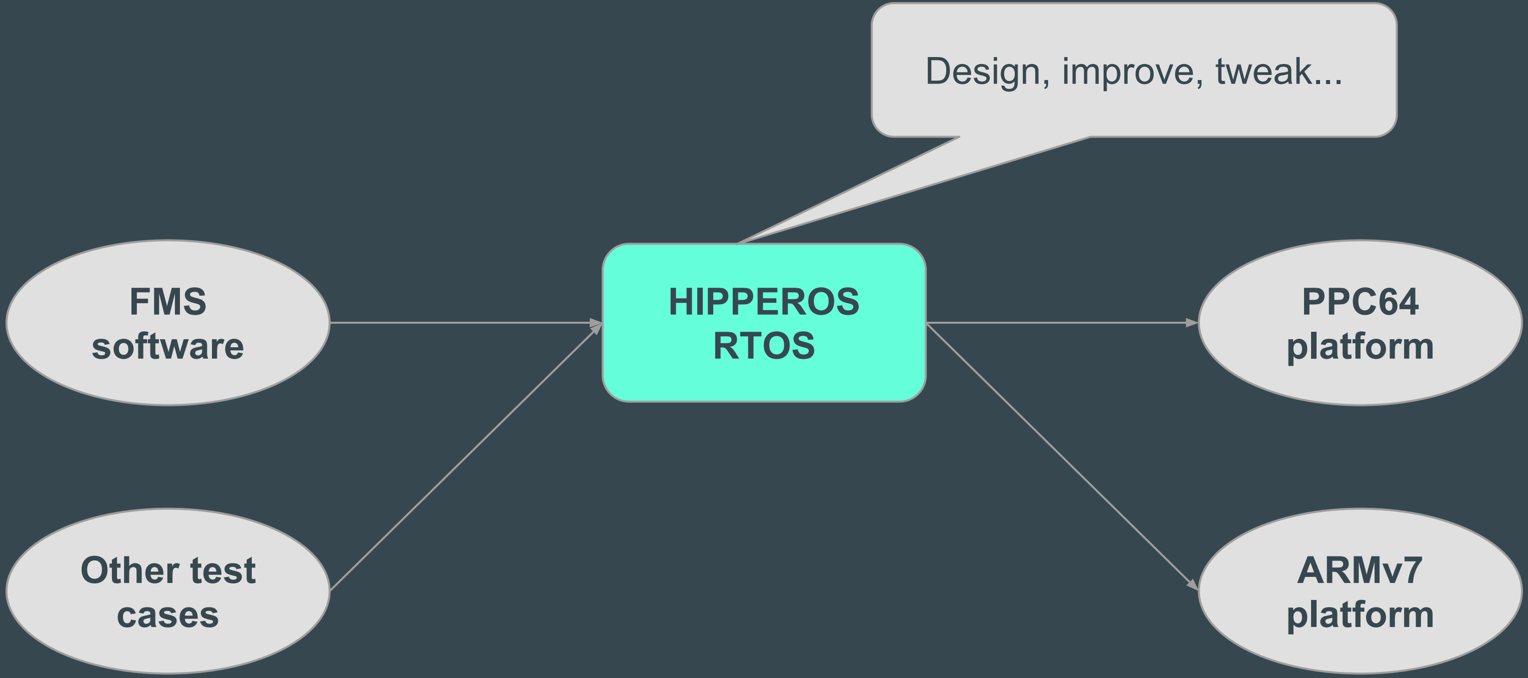
Possible solutions

Possible solutions

Possible solutions

Possible solutions

Possible solutions



Project roadmap

Project roadmap

1. Support of the PPC64 architecture for HIPPEROS

Project roadmap

1. Support of the PPC64 architecture for HIPPEROS

2. Port the FMS application on HIPPEROS

Project roadmap

1. Support of the PPC64 architecture for HIPPEROS

2. Port the FMS application on HIPPEROS

3. Extend HIPPEROS with Mixed-Criticality scheduling capabilities

Project roadmap

1. Support of the PPC64 architecture for HIPPEROS

2. Port the FMS application on HIPPEROS

3. Extend HIPPEROS with Mixed-Criticality scheduling capabilities

4. Validate the setup with experiments

Project roadmap

1. Support of the PPC64 architecture for HIPPEROS

2. Port the FMS application on HIPPEROS

3. Extend HIPPEROS with Mixed-Criticality scheduling capabilities

4. Validate the setup with experiments

Presentation Agenda

A. (Short) Presentation of the Thales application

B. Description of the RTOS

C. Description of the platform(s)

D. Experiments

A. The Application

The application use case

The application use case

- Provided by Thales as a case study

The application use case

- Provided by Thales as a case study

- “Autopilot”:
    - Inputs: sensors (GPS signal), database
    - Outputs: display, direction instructions

The application use case

- Provided by Thales as a case study

- "Autopilot":
  - Inputs: sensors (GPS signal), database
  - Outputs: display, direction instructions

- 18 cooperating high criticality tasks with different periodicity attributes

The application use case

- Provided by Thales as a case study

- "Autopilot":
  - Inputs: sensors (GPS signal), database
  - Outputs: display, direction instructions

- 18 cooperating high criticality tasks with different periodicity attributes

- Programming model: Acquisition - Execution - Restitution

The application use case

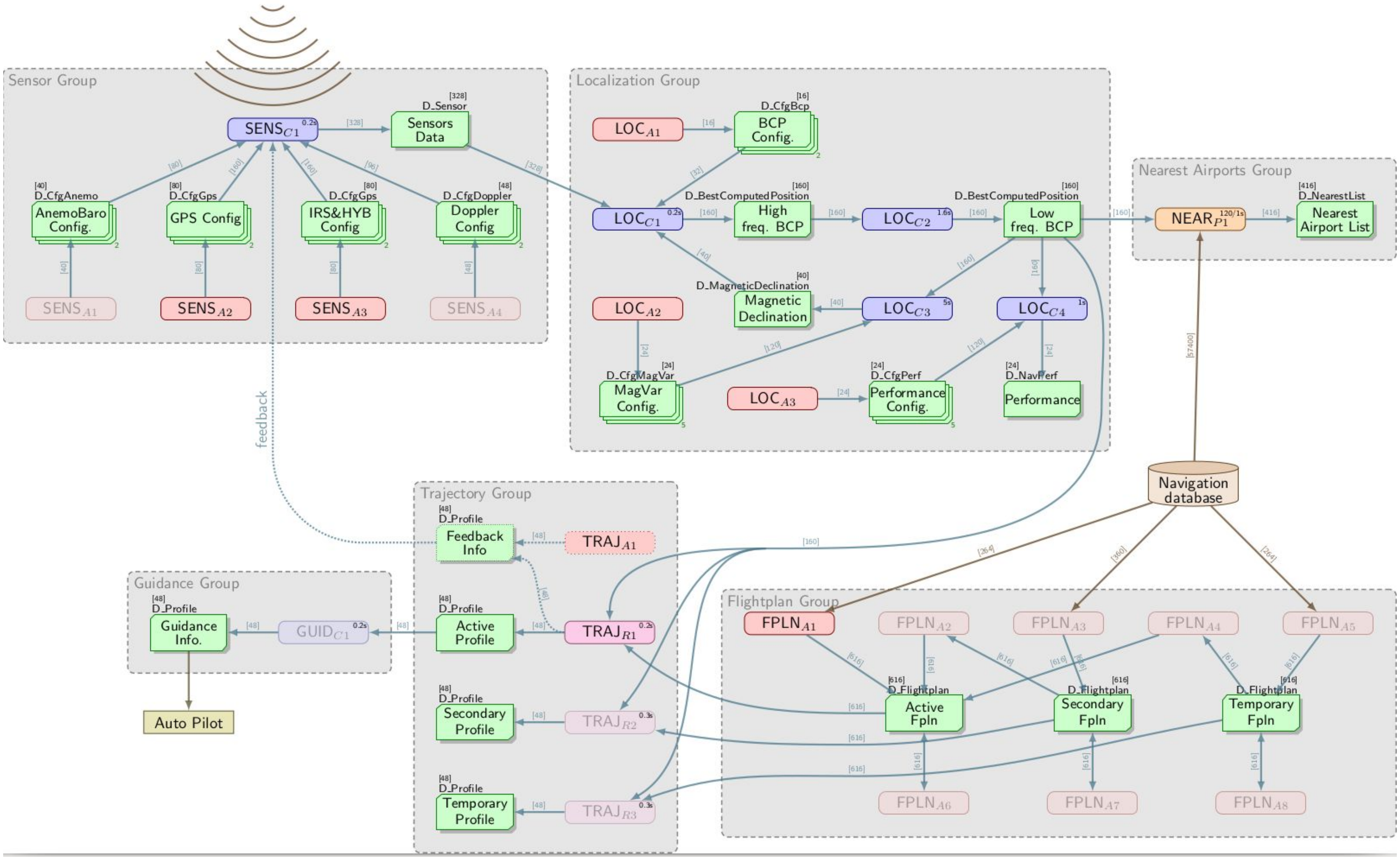
- Provided by Thales as a case study

- "Autopilot":
  - Inputs: sensors (GPS signal), database
  - Outputs: display, direction instructions

- 18 cooperating high criticality tasks with different periodicity attributes

- Programming model: Acquisition - Execution - Restitution

- The sampled use case takes 90 seconds to run



Sensor Group
SENS_C1
Sensors Data
D_Sensor
AnemoBaro Config.
GPS Config
IRS&HYB Config
Doppler Config
D_CfgAnemo
D_CfgGps
D_CfgGps
D_CfgDoppler
SENS_A1
SENS_A2
SENS_A3
SENS_A4
feedback
Localization Group
LOC_A1
BCP Config.
D_CfgBcp
LOC_C1
High freq. BCP
D_BestComputedPosition
LOC_C2
Low freq. BCP
D_BestComputedPosition
Magnetic Declination
D_MagneticDeclination
LOC_C3
LOC_C4
LOC_A2
MagVar Config.
D_CfgMagVar
LOC_A3
Performance Config.
D_CfgPerf
Performance
D_NavPerf
Nearest Airports Group
NEAR_P1
Nearest Airport List
D_NearestList
Navigation database
Trajectory Group
Feedback Info
TRAJ_A1
Active Profile
TRAJ_R1
Secondary Profile
TRAJ_R2
Temporary Profile
TRAJ_R3
D_Profile
Guidance Group
Guidance Info.
GUID_C1
Auto Pilot
Flightplan Group
FPLN_A1
FPLN_A2
FPLN_A3
FPLN_A4
FPLN_A5
Active Fpln
Secondary Fpln
Temporary Fpln
D_Flightplan
FPLN_A6
FPLN_A7
FPLN_A8

B. The Operating System

Context: the HIPPEROS company builds HIPPEROS OSes

- Development of the kernel started in June 2013
- Spin-off company, from Université Libre de Bruxelles, created in January 2014
- Today: ~15 people among them 5 OS developers & researchers
- The goal is to ship certifiable OSes to safety-critical software industries

Main architecture and design choices

Main architecture and design choices

- Hard real-time operating system

Main architecture and design choices

- Hard real-time operating system
- Embedded targets: ARMv7, ARMv8, IA32, PowerPC 64

Main architecture and design choices

- Hard real-time operating system
- Embedded targets: ARMv7, ARMv8, IA32, PowerPC 64
- A new micro-kernel written from scratch
  - Built for user needs, i.e. small footprint and adapted policies
  - Multi-core architecture based on an asymmetric kernel
  - Real-time model for user applications
  - MMU support and virtual address space
  - Resource sharing & IPC protocols (mutexes, semaphores, message passing, etc.)
  - Usual OS services (timers, etc.)

Main architecture and design choices

- Hard real-time operating system
- Embedded targets: ARMv7, ARMv8, IA32, PowerPC 64
- A new micro-kernel written from scratch
  - Built for user needs, i.e. small footprint and adapted policies
  - Multi-core architecture based on an asymmetric kernel
  - Real-time model for user applications
  - MMU support and virtual address space
  - Resource sharing & IPC protocols (mutexes, semaphores, message passing, etc.)
  - Usual OS services (timers, etc.)
- The OS is highly configurable

Real-time

- User processes have real-time requirements
- Determinism and bounded guarantees
- On time as opposed to fast
- Real-time scheduling policies
- Resource usage bounded and checked

New micro-kernel

New micro-kernel

- No “Linux legacy” or other previous mono-core design

New micro-kernel

- No “Linux legacy” or other previous mono-core design
- Design for SMP platforms

New micro-kernel

- No “Linux legacy” or other previous mono-core design

- Design for SMP platforms

- Asymmetric kernel design
  - One core for heavy scheduling operations
  - Other cores working to service tasks
  - Remote system call mechanism

New micro-kernel

- No “Linux legacy” or other previous mono-core design

- Design for SMP platforms

- Asymmetric kernel design
    - One core for heavy scheduling operations
    - Other cores working to service tasks
    - Remote system call mechanism

- Most services & drivers in user space

New micro-kernel

- No “Linux legacy” or other previous mono-core design

- Design for SMP platforms

- Asymmetric kernel design
    - One core for heavy scheduling operations
    - Other cores working to service tasks
    - Remote system call mechanism

- Most services & drivers in user space

- Multi-core IPC protocol to handle it

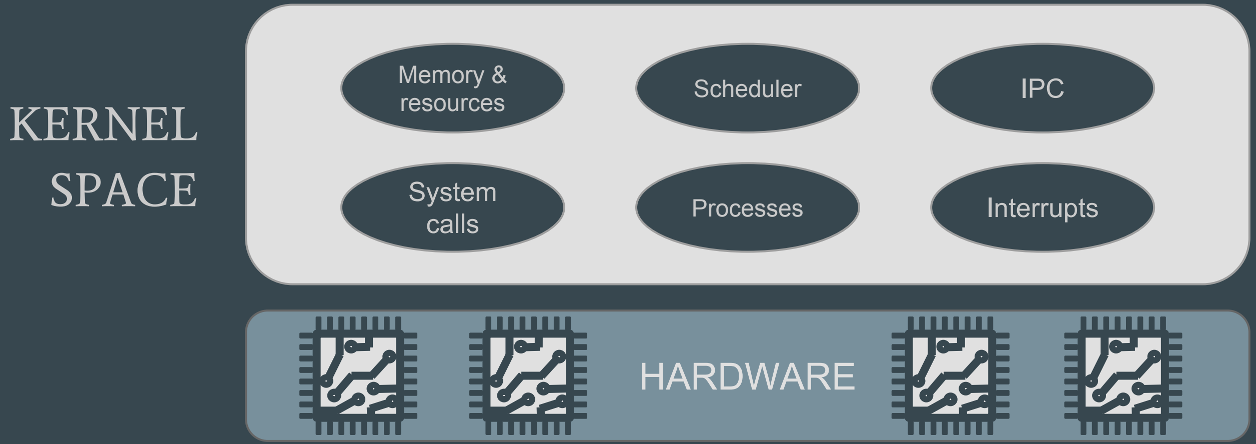
New micro-kernel

- No “Linux legacy” or other previous mono-core design
- Design for SMP platforms
- Asymmetric kernel design
    - One core for heavy scheduling operations
    - Other cores working to service tasks
    - Remote system call mechanism
- Most services & drivers in user space
- Multi-core IPC protocol to handle it

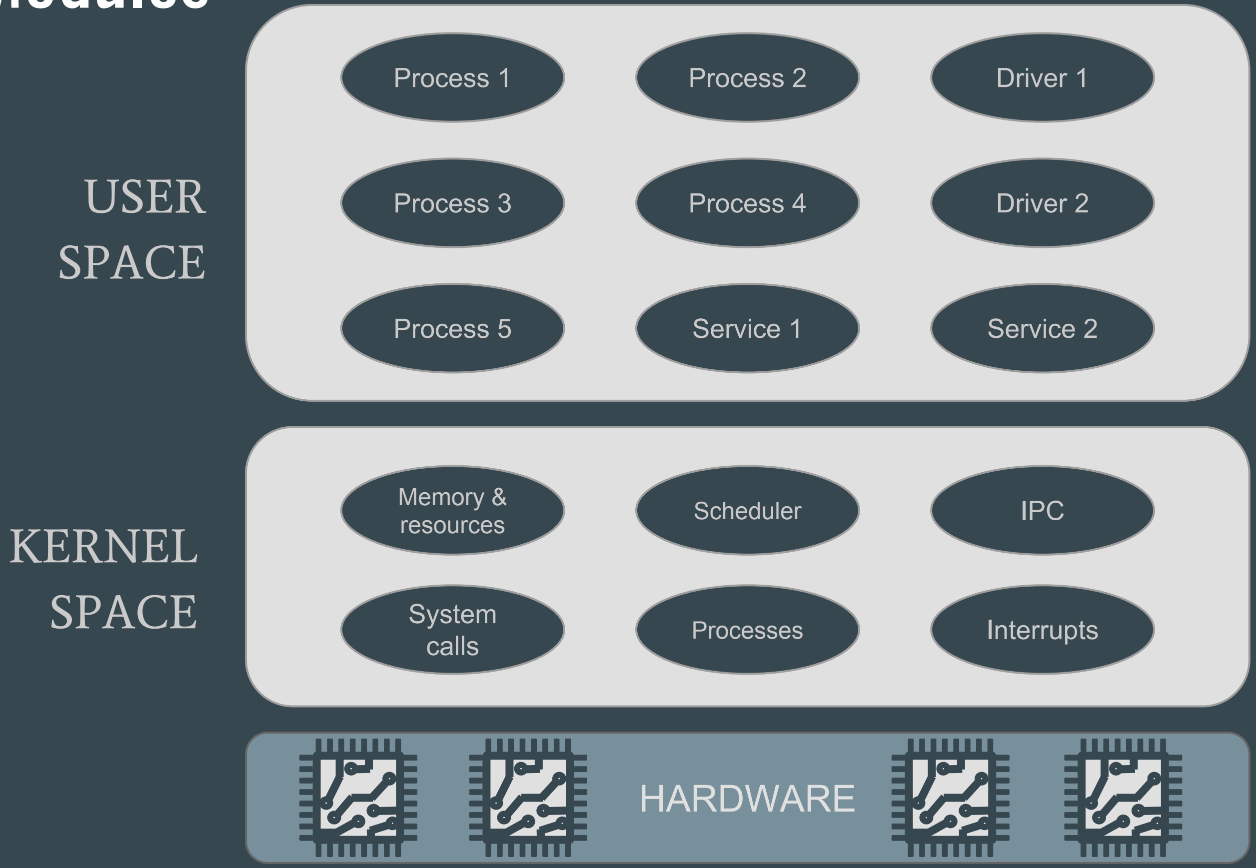
NB: we also support FPGA :-)

OS Modules

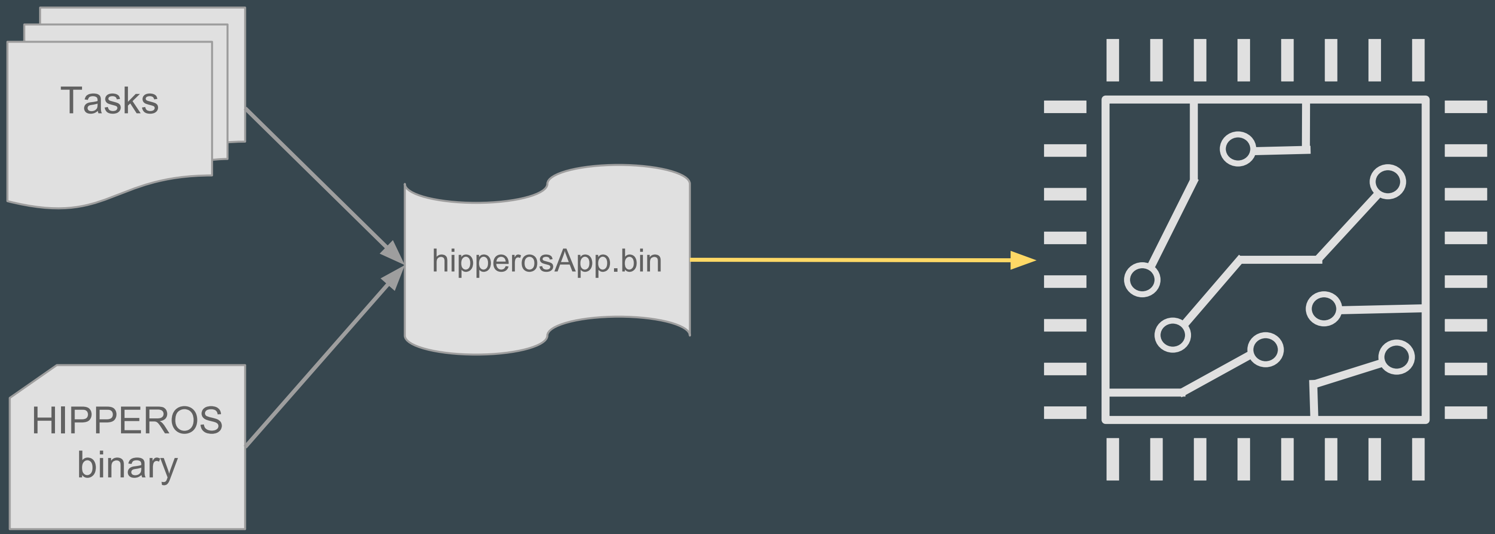
OS Modules


KERNEL
SPACE
Memory & resources
Scheduler
IPC
System calls
Processes
Interrupts
HARDWARE

OS Modules


USER
SPACE
Process 1
Process 2
Driver 1
Process 3
Process 4
Driver 2
Process 5
Service 1
Service 2
KERNEL
SPACE
Memory & resources
Scheduler
IPC
System calls
Processes
Interrupts
HARDWARE

In practice: build an application and deploy it on target



Task set defines real-time behaviour and specification

- Timing parameters
- Periodicity
- Code
- Core affinities
- …

```xml
<task>
  <identifier>1</identifier>
  <name>task1</name>
  <stackSize>8192</stackSize>
  <recurrence>UNIQUE</recurrence>
  <entryPoint>task1_main</entryPoint>
  <coreAffinity>0 1</coreAffinity>
</task>
<task>
  <identifier>2</identifier>
  <name>task2</name>
  <stackSize>8192</stackSize>
  <recurrence>PERIODIC</recurrence>
  <timingInformation>
    <wcet>42000</wcet>
    <deadline>20000</deadline>
    <period>20000</period>
  </timingInformation>
  <flags>REALTIME</flags>
  <coreAffinity>0</coreAffinity>
</task>
<task>
  <identifier>3</identifier>
  <name>task3</name>
  <stackSize>8192</stackSize>
  <recurrence>SPORADIC</recurrence>
  <timingInformation>
    <offset>10000</offset>
    <wcet>57000</wcet>
    <deadline>120000</deadline>
    <period>120000</period>
  </timingInformation>
  <flags>REALTIME</flags>
  <coreAffinity>1</coreAffinity>
</task>
```

For more information

- Seminal paper: OSPERT 15
- We can work together
  → HIPPEROS Academic Partner Program
  → academic@hipperos.com
- Use HIPPEROS for commercial application
  → contact us: info@hipperos.com
- You will soon be able to play with it for free!
  → HIPPEROS community edition
  → Expected release date: mid 2018
- For any information, contact me: antonio.paolillo@hipperos.com

A New Configurable and Parallel Embedded Real-time Micro-Kernel for Multi-core platforms

Antonio Paolillo, Olivier Desenfans, Vladimir Svoboda, Joël Goossens, Ben Rodriguez

PARTS Research Center, Université Libre de Bruxelles, Mangogem S.A. and HIPPEROS S.A.
Corresponding author: antonio.paolillo@ulb.ac.be

***Abstract*—One of the main on-going initiatives of the *PARTS Research Center* together with *HIPPEROS S.A.* is the creation of a new Real-Time Operating Systems family called *HIPPEROS*. This paper focuses on the design and the implementation of its new real-time multi-core micro-kernel. It aims to address the challenge of efficient management of computing resources for competing real-time workloads on modern MPSoC platforms while maintaining the level of assurance and reliability of existing production systems. The objective of this paper is to present an overview of its inner architecture.**

I. Introduction

For the past twenty years, real-time theory has widely explored the possibility to use multi-core and many-core platforms for embedded systems. However, while this topic seems to be very mature in the literature, the safety-critical software industry still relies on uni-core techniques for operating system implementations. The industry state-of-the-art regarding multi-core platforms is to fully separate the processing resources in time and space isolated partitions with very few possible communication channels between the different partitions and therefore to consider that each partition operates as an independent uni-core platform. Examples of de facto standards for these techniques are ARINC653 [1] and AUTOSAR [2].

While from a conservative point of view these are the most reliable and predictable solutions, these are not the best available options in the real-time research w.r.t. efficiency, resource utilisation and cost. Moreover, the rising demand of computational power per silicon area happening in every domain — including safety-critical systems — puts a pressure on the low-level, middleware and kernel developers to implement efficient policies for real-time process management.

In recent years, efforts in the research community have been made to adapt existing general-purpose kernels such as Linux in order to provide a real-time execution environment suitable for the evaluation of efficient multi-core real-time scheduling and resource allocation policies [3], [4]. The main advantage of this approach is to reuse the existing kernel code base which has been tested and validated by millions of users worldwide. However this approach cannot be directly exploited in production systems. Indeed Linux is not originally intended nor designed to support neither *hard* real-time constraints nor safety-critical applications. Moreover it is not conforming to the highest demanding certification standards of this industry such as DO-178-B (level A, B) or ISO26262. As a consequence, the latest real-time multi-core algorithms have not been tested in a strict and realistic hard real-time environment yet. As stated by Brandenburg in [5]:

> Ideally, [...], worst-case kernel overheads [...] should be determined analytically. However, for the foreseeable future, this will likely not be possible in complex kernels such as Linux. Instead, it would be beneficial to develop (or extend existing) μ-kernels of much simpler design with LITMUS^{RT}-like functionality.

This kind of kernel would have to be built from the ground up with hard real-time and multi-core constraints integrated as parts of its base design principles. This would allow for simpler, finer-grained measurements of the overheads introduced by different implementations of the various solutions the literature has to offer. Moreover the architecture of this kernel must scale with an increasing number of cores to allow execution on many-core platforms.

In order to address the challenge of providing efficient multi-core kernel implementations while still providing the same level of assurance and reliability of the existing industry quality standards, the *PARTS Research Center*, together with the company *MangoGem S.A.*, launched the *HIPPEROS* project in 2010. The development of the *HIPPEROS* kernel started in June 2013. *HIPPEROS* aims to provide a family of RTOS solutions, each adapted specifically to the different needs of the real-time system designer and including the implementation of the latest results of the research community. It stands for *HIgh Performance Parallel Embedded Real-time Operating Systems*.

II. System overview

We started the project by developing a new kernel from scratch running as a bare-metal system on ARM and x86 systems. The objective is to have a fully configurable kernel, running transparently on different architectures and platforms with an arbitrary number of cores, that will be the seed of the different RTOS solutions mentioned above. With such a flexible design it would be possible to deeply explore the practicability of real-time theory solutions. To reach this the kernel has the following design characteristics:

- for scalability reasons, it has a distributed asymmetric micro-kernel architecture, meaning that each core can execute a local part of the kernel (the lightweight and very local operations like simple system calls or process context switching), while a dedicated core executes the heavy parts of the kernel (complex system calls, scheduling decisions, shared resources handling, etc), allowing to execute several parts of the kernel *in parallel*; to the

25

Mixed-criticality operating system?

Mixed-criticality operating system?

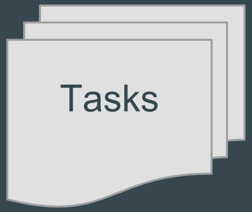
Vestal model

Mixed-criticality operating system?

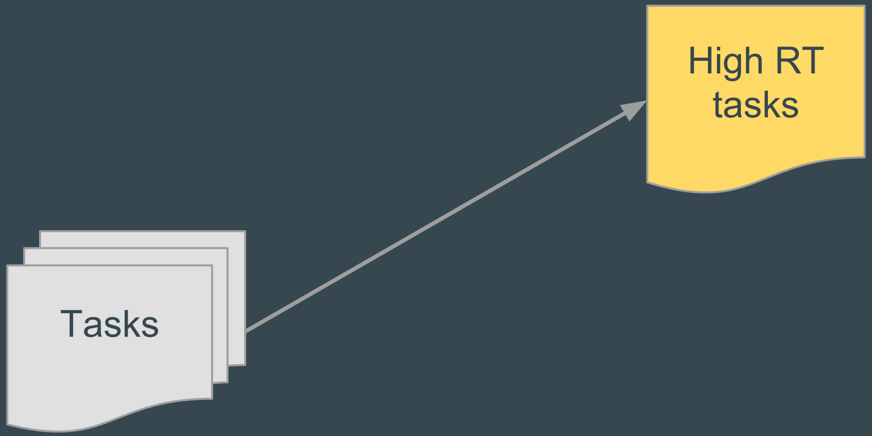
Vestal model*

* actually, the elastic task model

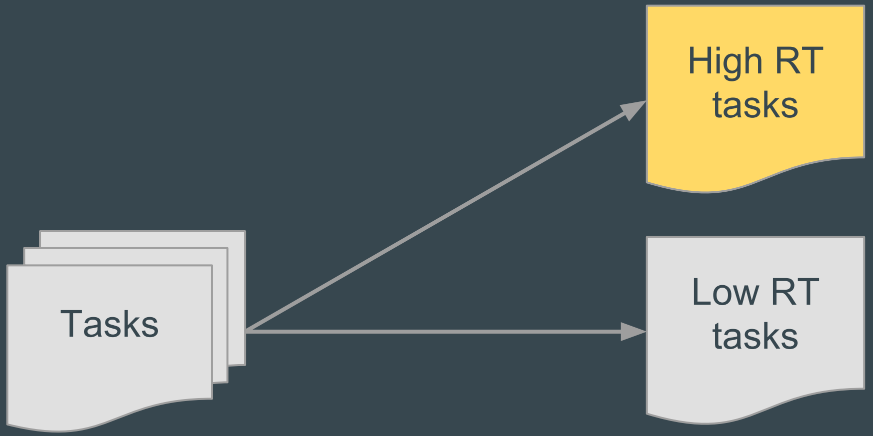
3 types of tasks



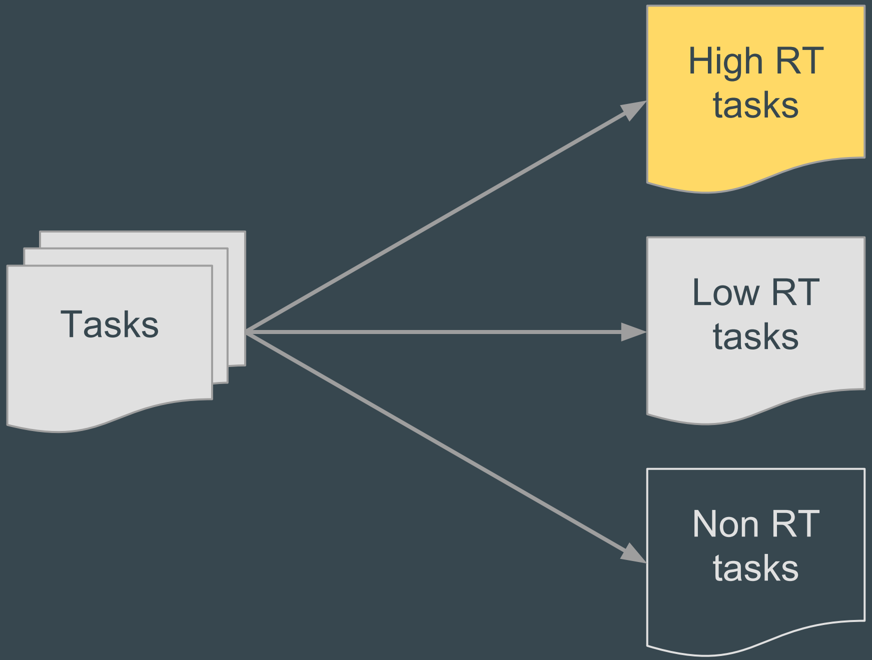
3 types of tasks



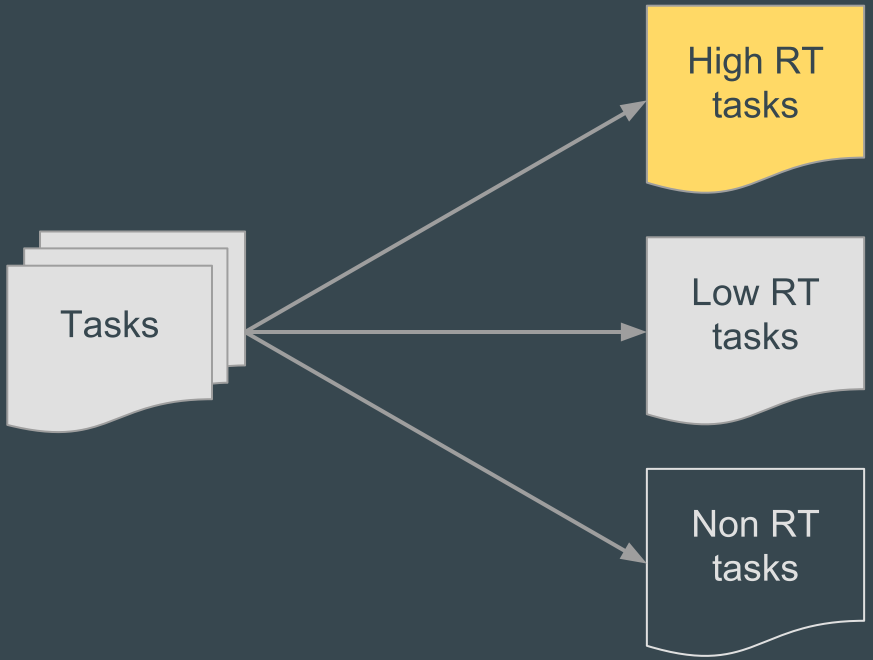
3 types of tasks



3 types of tasks

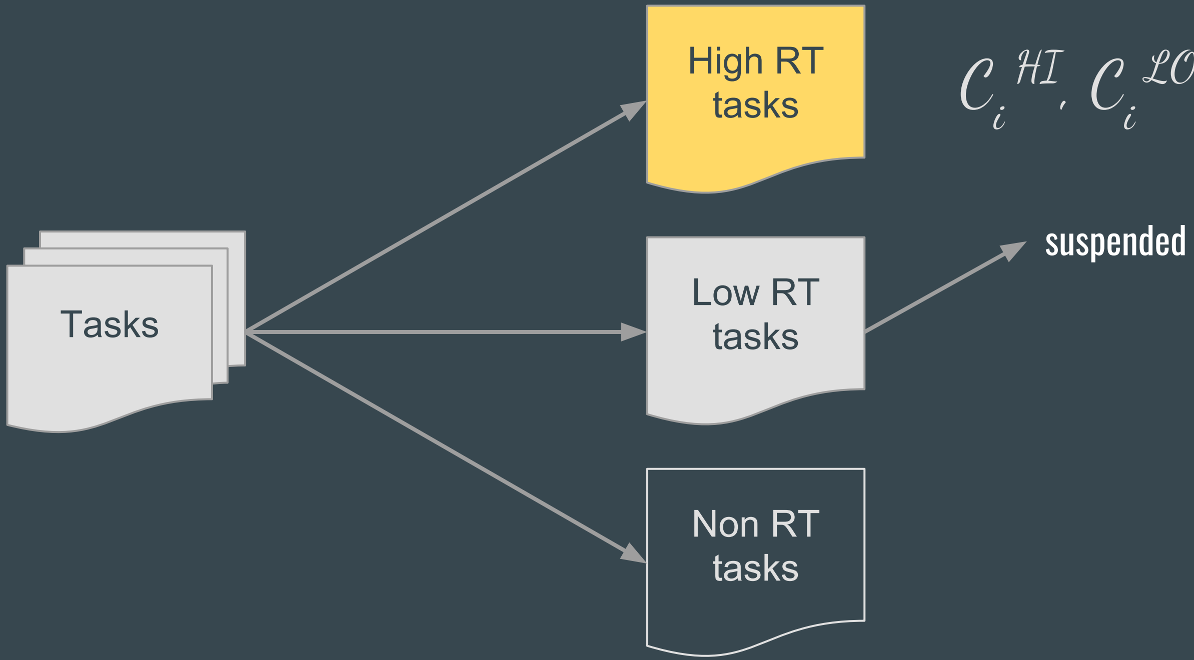


3 types of tasks



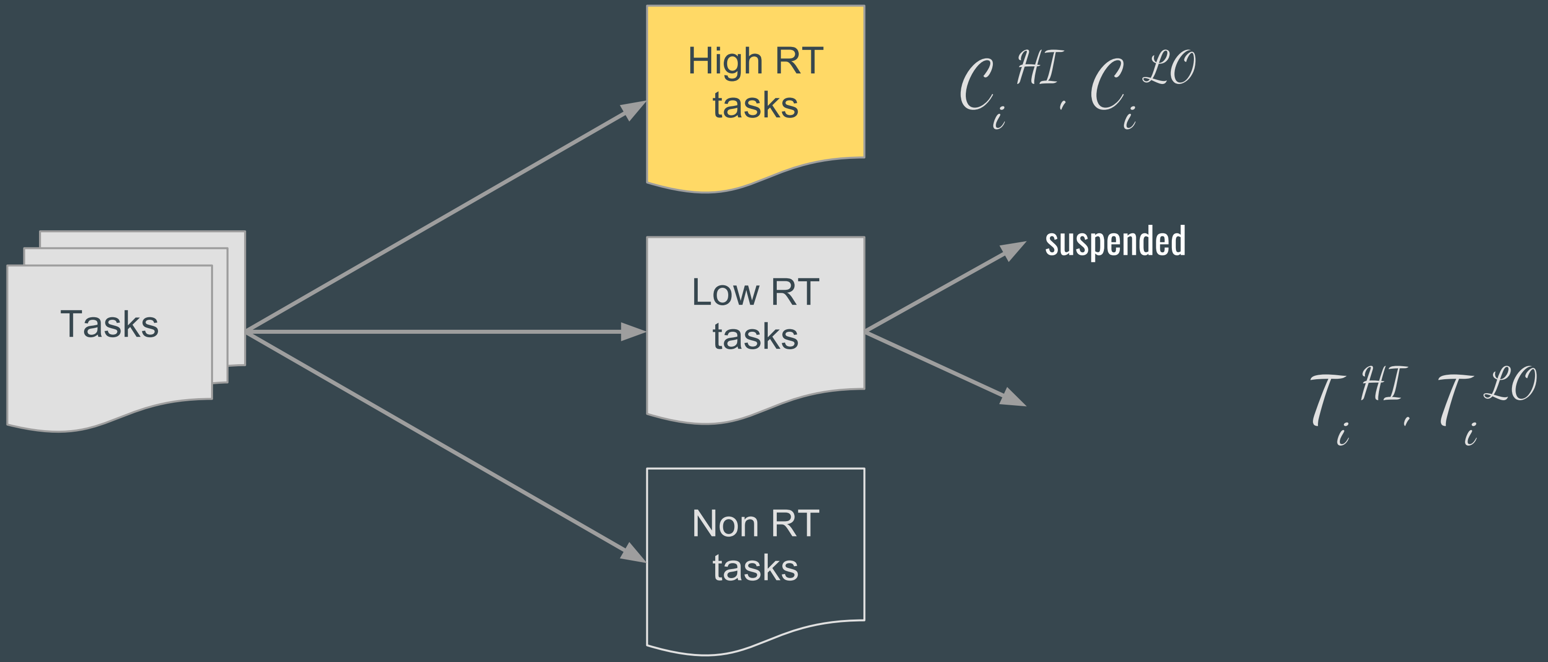
C_i^{HI}, C_i^{LO}

3 types of tasks

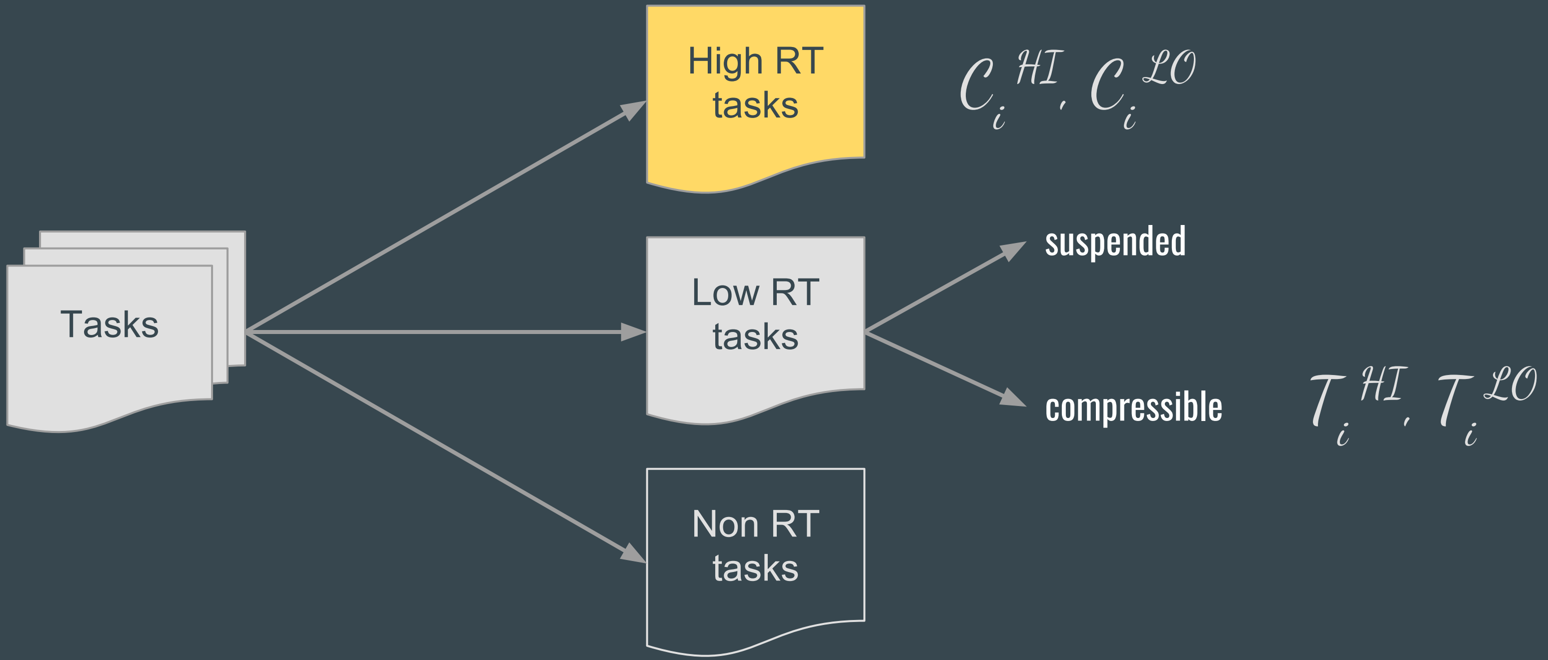


C_i^{HI}, C_i^{LO}

3 types of tasks



3 types of tasks



Extend the task set with Mixed-Criticality

```
<task>
  <timingInformation>
    <offset>0</offset>
    <wcet>100000</wcet>
    <deadline>200000</deadline>
    <period>500000</period>
  </timingInformation>
</task>
```

Extend the task set with Mixed-Criticality

```xml
<task>
  <timingInformation>
    <offset>0</offset>
    <wcet>100000</wcet>
    <deadline>200000</deadline>
    <period>500000</period>
  </timingInformation>
</task>
```

$\langle O_i, C_i, D_i, T_i \rangle$

Extend the task set with Mixed-Criticality

```xml
<task>
  <timingInformation>
    <offset>0</offset>
    <wcet>100000</wcet>
    <deadline>200000</deadline>
    <period>500000</period>
    <mcHigh>
      <wcetLow>25000</wcetLow>
    </mcHigh>
  </timingInformation>
</task>
```

$$\langle O_i, C_i^{HI}, D_i, T_i \rangle$$

$$C_i^{LO}$$

Extend the task set with Mixed-Criticality

```xml
<task>
  <timingInformation>
    <offset>0</offset>
    <wcet>100000</wcet>
    <deadline>200000</deadline>
    <period>500000</period>
    <mcLow>
        <periodHigh>1000000</periodHigh>
        <onModeSwitch>LET_FINISH</onModeSwitch>
    </mcLow>
  </timingInformation>
</task>
```

$$\langle O_i, C_i, D_i, T_i^{LO} \rangle$$

$$T_i^{HI}$$

Mode switch event - a low WCET overrun

Low tasks job instance:

```
<onModeSwitch>
  LET_FINISH
</onModeSwitch>
```

OR

```
<onModeSwitch>
  KILL
</onModeSwitch>
```

Tasks:

```
<periodHigh>
  1000000
</periodHigh>
```

OR

```
<suspended/>
```

OR

```
<unaffected/>
```

HI task LO WCET overrun?

HI task LO WCET overrun?

Global mode switch

(all jobs of all cores)

Switch back to LO mode?

Switch back to LO mode?

First idle instant

C. Evaluated platforms

NXP T2080RDB

- QorIQ T2080 platform:
  - 4 dual-threaded e6500 cores (1.8 GHz)
  - PowerPC 64 bits architecture

- 4 GB RAM

- OS support for caches:
  - L1
  - Partitioning of L2 (≃ private)
  - No L3
  - TLB miss software handler

BD SABRE Lite

- NXP i.MX 6Quad processor:
    - 4 Cortex-A9 cores (800 MHz)
    - ARMv7-A 32 bits architecture
- 1 GB RAM
- OS support for caches:
    - Private L1 enabled
    - No L2
- Not in IMICRASAR but “control board”

D. Experiments

What we evaluated

What we evaluated

- Idea: how to set the LO wcet such that we balance safety and efficiency?

What we evaluated

- Idea: how to set the LO wcet such that we balance safety and efficiency?

- Experiment execution time measurements → approximate HI WCET bounds
    - HI WCET: 4x max observed execution time for each task

What we evaluated

- Idea: how to set the LO wcet such that we balance safety and efficiency?

- Experiment execution time measurements → approximate HI WCET bounds
  - HI WCET: 4x max observed execution time for each task

- Run the Thales HI application concurrently with a dummy LO application

What we evaluated

- Idea: how to set the LO wcet such that we balance safety and efficiency?

- Experiment execution time measurements → approximate HI WCET bounds
  - HI WCET: 4x max observed execution time for each task

- Run the Thales HI application concurrently with a dummy LO application
  - Add dummy LO tasks and measure contributions
  - CPU bound, no I/O
  - Suspended when a mode switch occurs
  - No deadline miss for HI tasks (avoid HI/LO interferences)

What we evaluated

- Idea: how to set the LO wcet such that we balance safety and efficiency?

- Experiment execution time measurements → approximate HI WCET bounds
    - HI WCET: 4x max observed execution time for each task

- Run the Thales HI application concurrently with a dummy LO application
    - Add dummy LO tasks and measure contributions
    - CPU bound, no I/O
    - Suspended when a mode switch occurs
    - No deadline miss for HI tasks (avoid HI/LO interferences)

- Limitations:
    - partitioned fixed priority scheduling
    - no specific MC scheduler is implemented (future work)

Summary: evaluation scheme

A. Measure job execution time to bound WCET

B. Run the HI use case with a LO application

Summary: evaluation scheme

A. Measure job execution time to bound WCET

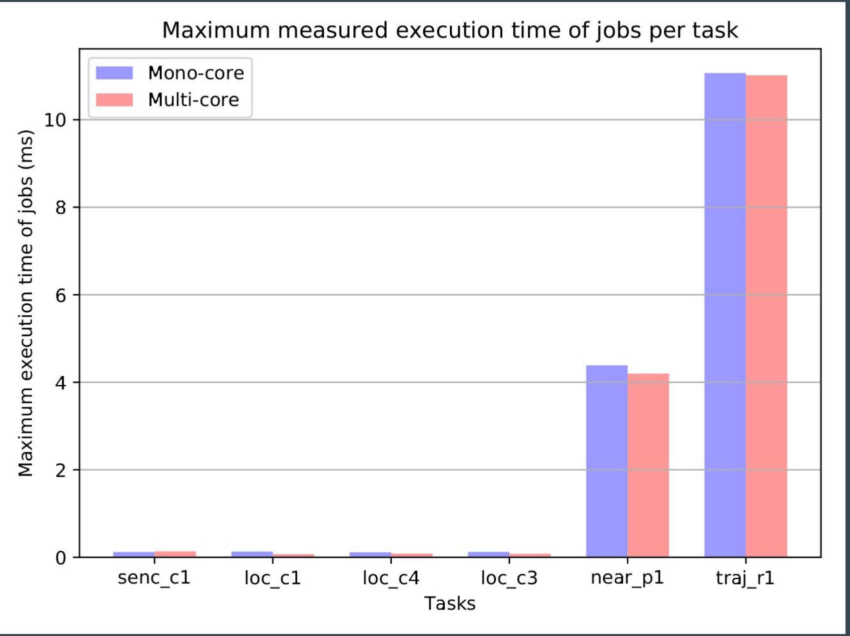
B. Run the HI use case with a LO application

Job execution time measurements

- Thales FMS application: 18 tasks
- 12 tasks have negligible execution time and/or are not periodic
- We plot the maximum observed execution time of all the jobs of each task
- Repeated 10 times, very stable measurements (small stdev)

Job execution time measurements (max observed)

T2080RDB



SABRE Lite

Observation

Use case uses less than 10%
of the platform CPU

Summary: evaluation scheme

A. Measure job execution time to bound WCET

B. Run the HI use case with a LO application

How to set WCET LO?

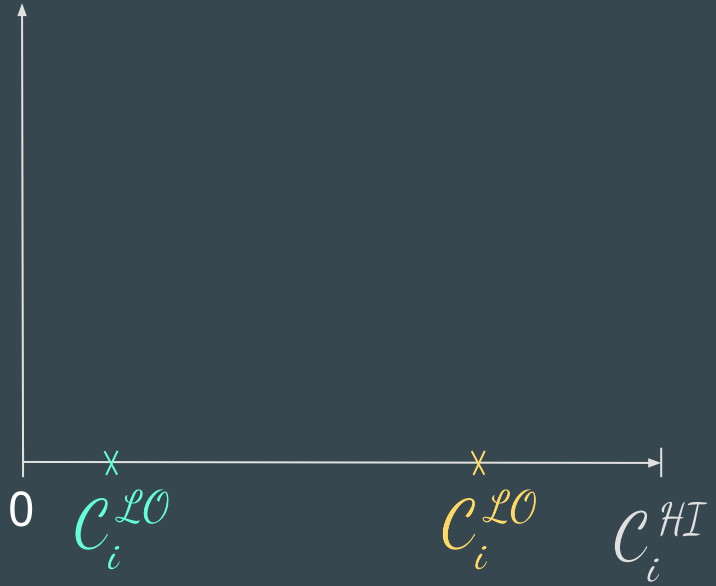
How to set WCET LO?

How to set WCET LO?

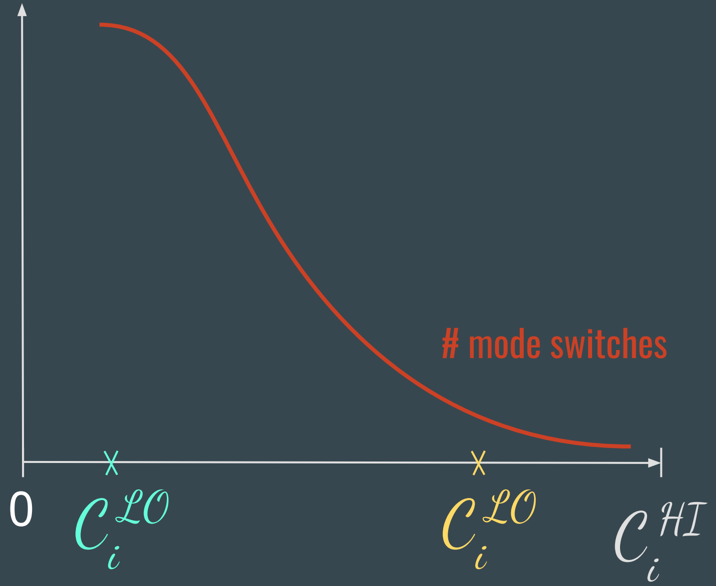
How to set WCET LO?

How to set WCET LO?

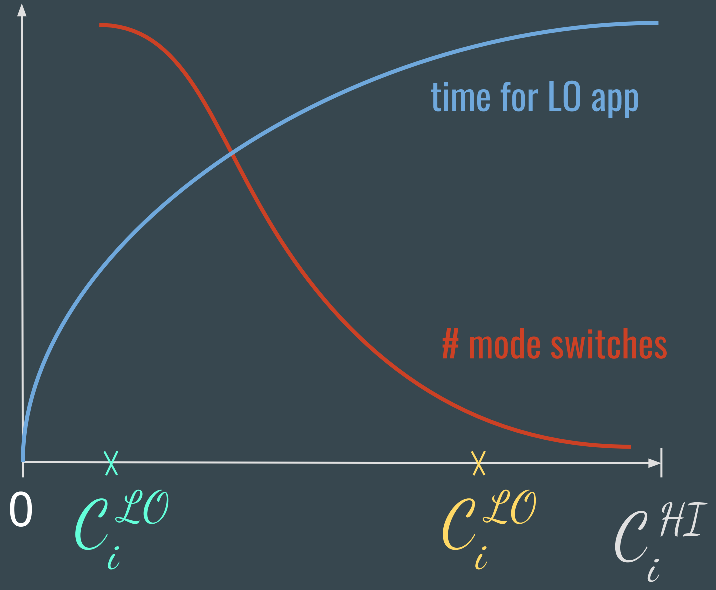
How to set WCET LO?



How to set WCET LO?

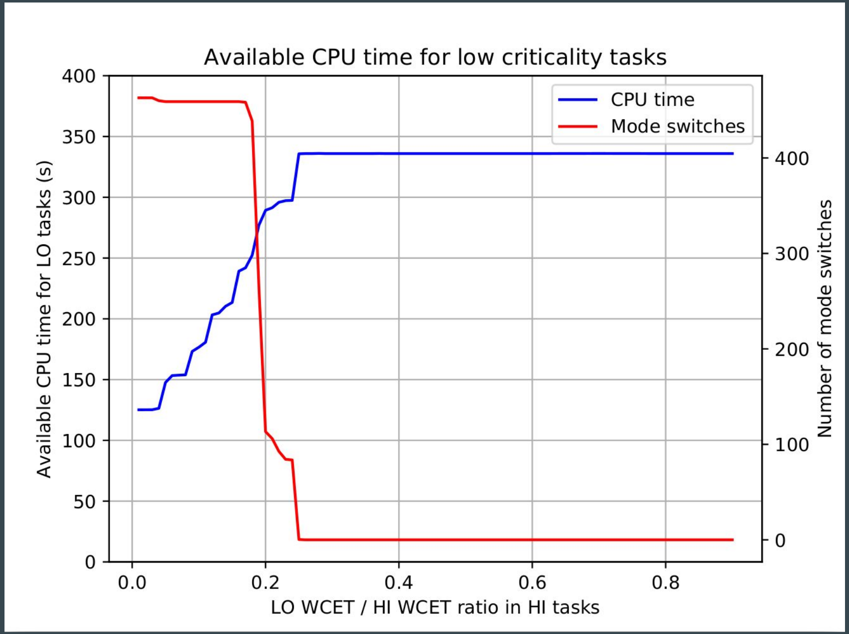


How to set WCET LO?

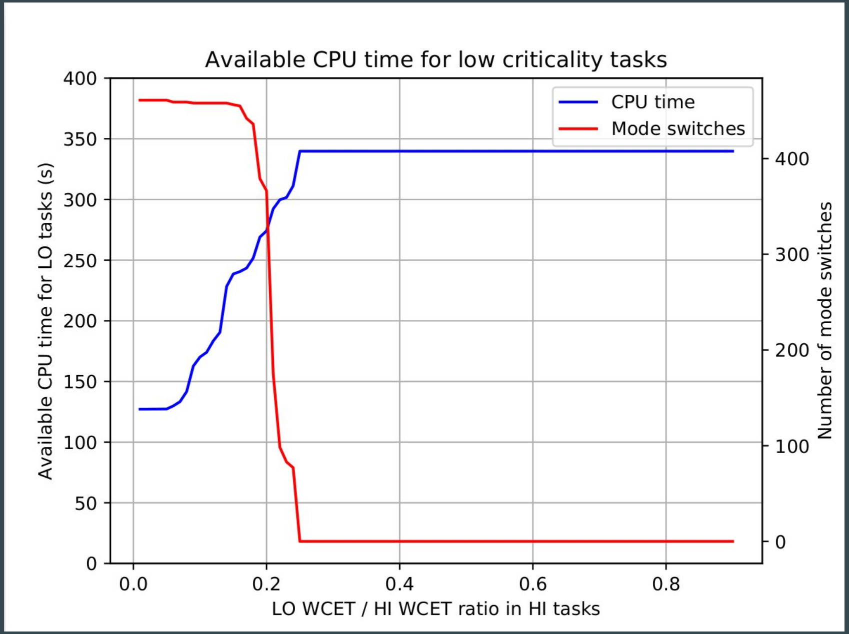


Evolution of available LO CPU time and # mode switches

T2080RDB

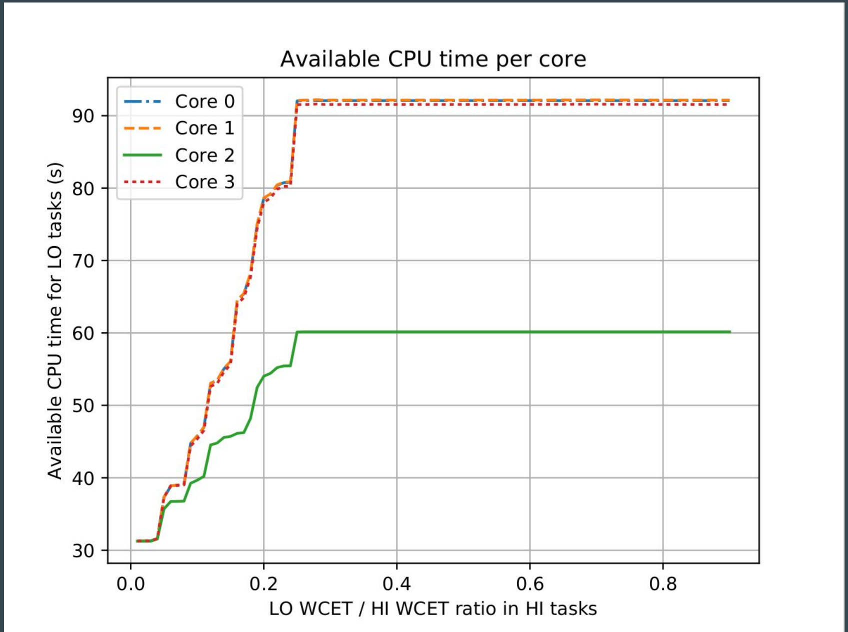


SABRE Lite

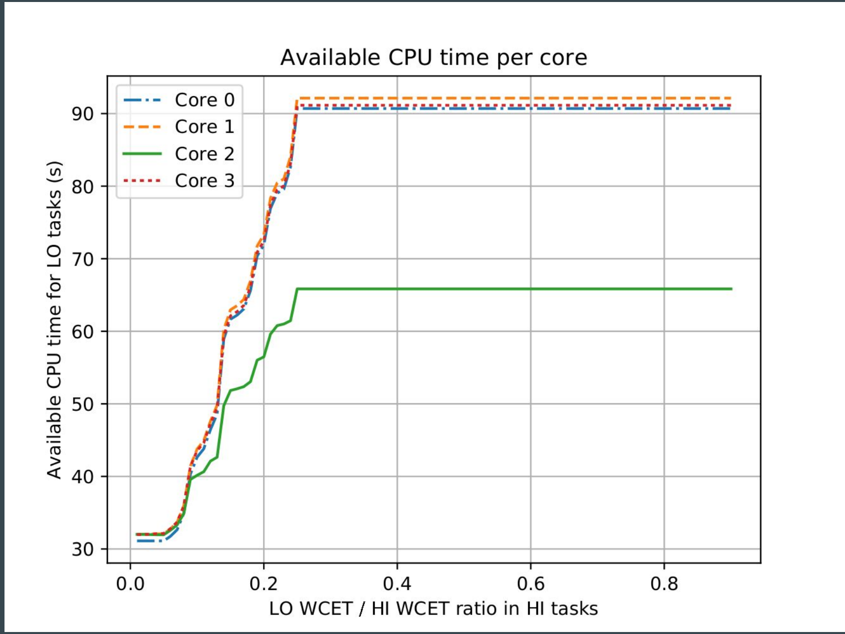


Evolution of available LO CPU time, per core

T2080RDB



SABRE Lite

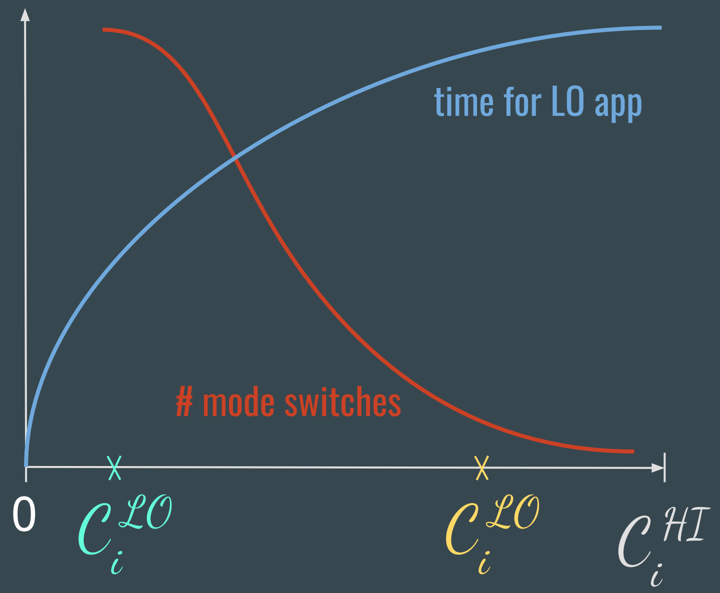


Conclusions

Conclusions

- About the porting : no line of the original core application were modified
- Industrial use case takes less than 10% on the considered platform
- We can use the remaining 90% for LO app without compromising HI app
- Straightforward MC mode switch support at OS level
- Future work involves:
  - Fine grained control over interferences
  - Implementation of MC scheduling algorithms
  - Evaluations: non trivial LO app (memory, file systems, I/O) with compressible/incompressible
  - Typical industrial use cases

Where is the balance?



Where is the balance?