17th ESA Workshop on Avionics, Data, Control and Software Systems, ESTEC, 15th November 2023

PLATO N-DPU ASW dual-core architecture and the V&V approach followed

Philippe Plasson, philippe.plasson@obspm.fr

LESIA l'Observatoire de Paris | PSL

Laboratoire d'Études Spatiales et d'Instrumentation en Astrophysique

cnrs

SORBONNE UNIVERSITÉ

Université Paris Cité

PSL UNIVERSITÉ PARIS

Contents

PLATO N-DPU ASW dual-core architecture and the V&V approach followed

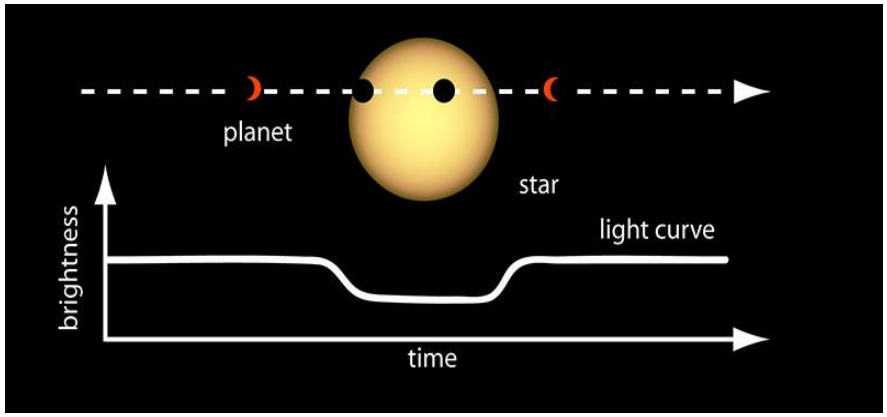
PLATO Overview

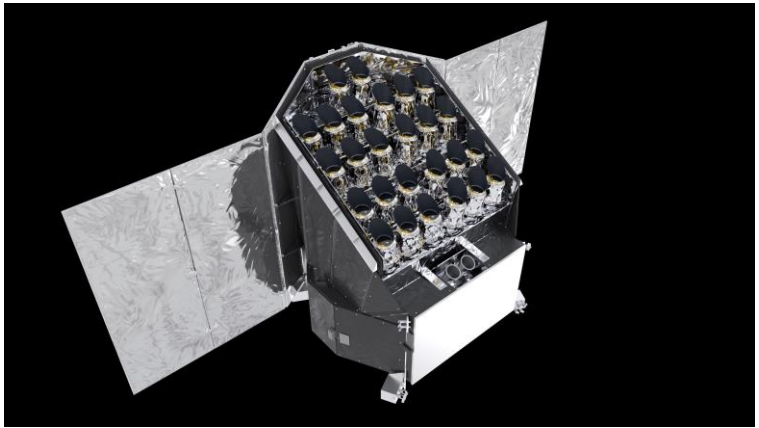
PLATO Mission

- PLATO = "PLAnetary Transits and Oscillations of stars"
- Third medium-class mission of the European Space Agency Cosmic Vision programme.
- Main objective = to detect terrestrial exoplanets in the habitable zone of Sun-like stars.
- Launch = end 2026.

PLATO Payload

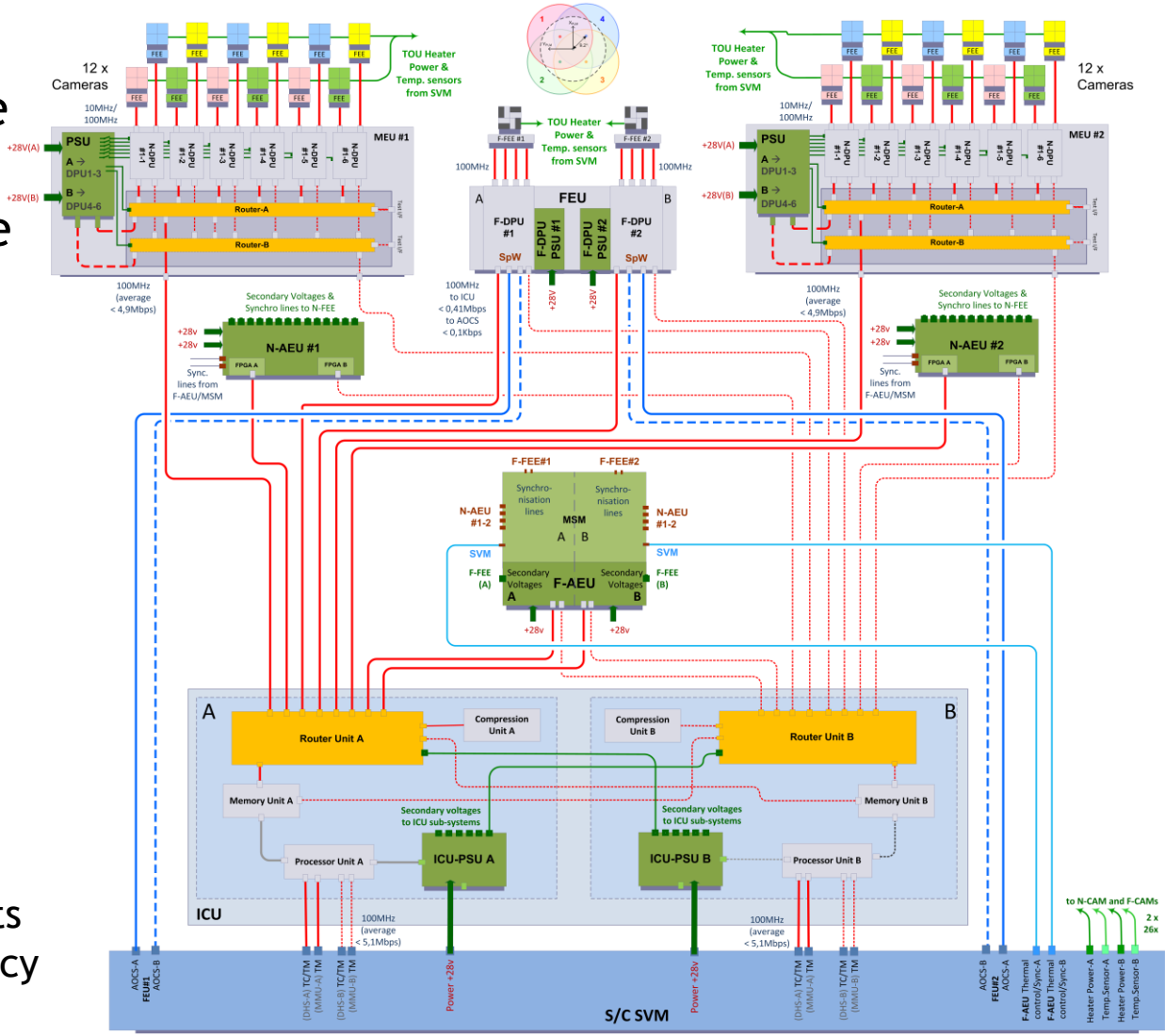
- Ultra-high precision, long, uninterrupted photometric monitoring in the visible of very large samples of bright stars
- Multi-camera approach: set of 26 cameras (2 gigapixels).
- Huge amount of data to process on-board:
  - 14 TB of data are generated each day by the 26 cameras
  - Only 54 GB can be downloaded to the ground
- From the start of the project, the question of the computing power needed on board was identified as critical.





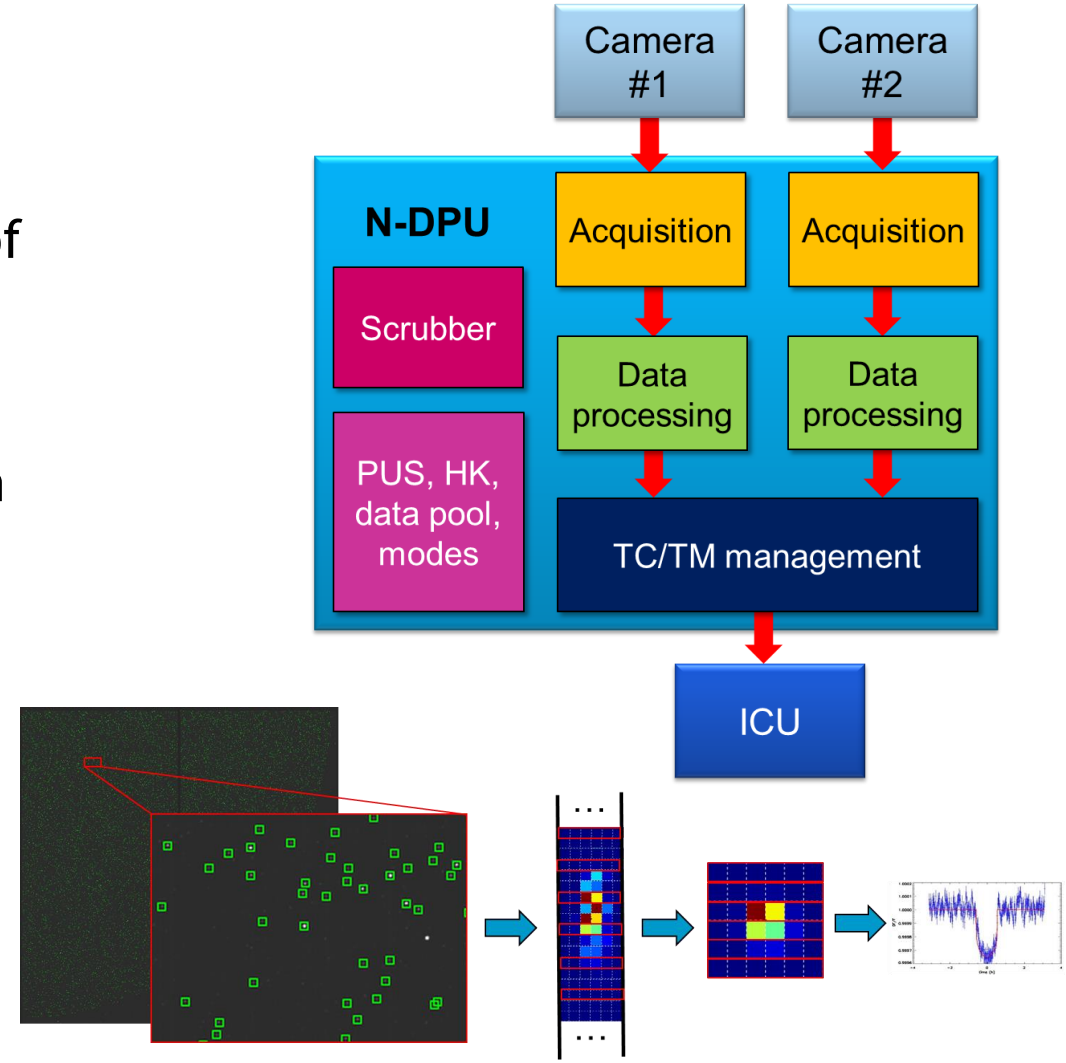
PLATO On-board Data Processing System

- The PLATO Data Processing System is the sub-system of PLATO payload in charge of the on-board data processing.
- 16 on-board computer units connected via a SpaceWire network:
  - 12 Normal Data Processing Units embedding a **dual-core LEON3-FT processor**
  - 2 Fast Data processing Units
  - 2 Instrument Control Units working in cold redundancy



PLATO N-DPU Application Software

- The N-DPU Application Software is the embedded software deployed in each of the 12 N-DPU boards.
- Each software manages two cameras by reading pixels and housekeeping data from the camera front-end electronics.
- The software must process up to 260000 stars every 25 seconds.
  - For 20% of the processed stars, the software produces 6x6-pixel windows.
  - For 80% of the stars, the software computes photometry products (fluxes, centers of brightness).



CCD read-out by N-FEE. Windowing is applied by N-FEE.

Segments of windows transmitted by N-FEE to N-DPU

Windows reconstructed by N-DPU

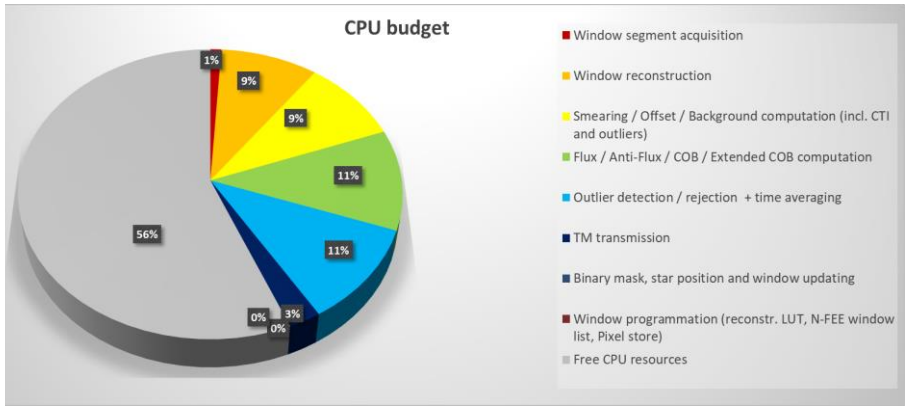
Fluxes and COB calculated by N-DPU

Contents

PLATO N-DPU ASW dual-core architecture and the V&V approach followed

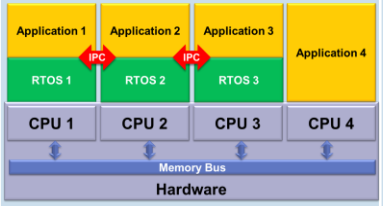
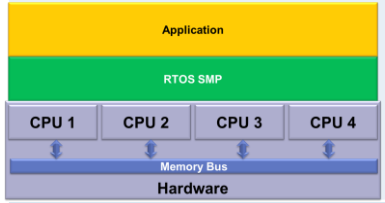
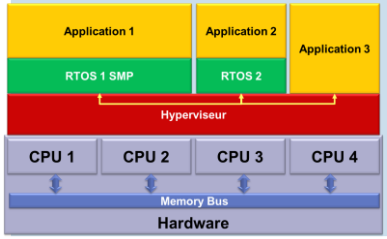
Why a dual-core architecture?

- The PLATO project was identified from the beginning as a challenge in terms of the data processing resources needed on board.
- We carried out in-depth assessment of the CPU budgets from the early phases of the project.
- CPU budgets established by prototyping in C the photometry algorithms and by measuring the execution times on a LEON processor simulator.



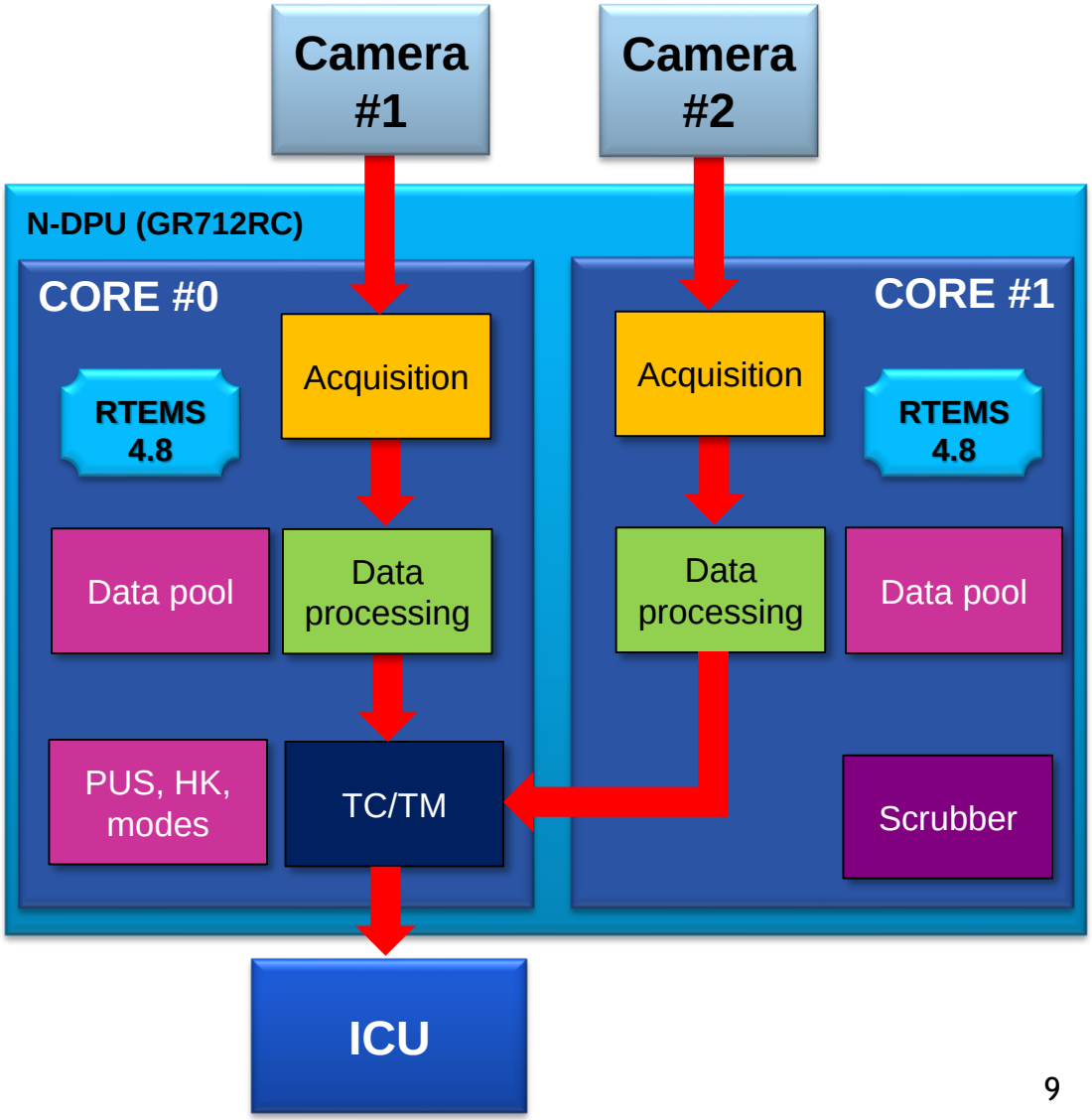
- For processing the data of one camera, the occupancy rate of a LEON processor operating at 50 MHz has been estimated at around 40-50%.
- Constraints in terms of weight, size and power consumption led to the decision that a DPU should manage 2 cameras.
- Solution adopted: **12 N-DPU with a LEON dual-core processor running at 50 MHz**

What type of multi-core architecture?

Approach	Main features / Pos and Cons
AMP (Asymmetric Multi-Processing) ✓ 	• Each core has its own OS and executes a separate set of tasks. • Optimizes resource utilization and minimizes the need for inter-core communication. • Adds the overhead of multiple OS instances. • Lack of an integrated process and tools to easily develop this type of architecture.
SMP (Symmetric Multi-Processing) ✗ 	• All cores share the same OS and communicate with each other via shared memory. • Provides a simpler programming model, but the execution model is generally more complex. • Rejected due to lack of a qualified RTOS supporting SMP (when the study was done)
Hypervisor ✗ 	• Infrastructure for implementing time and space partitioning in a multi-core system by virtualizing the physical resources, such as processing cores, memory, and I/O devices. • Rejected mainly because of a lack of expertise in this technology in the team and a lack of time to acquire the necessary know-how.

Why the AMP approach?

- Compatible with the use of qualified RTOS like RTEMS 4.8 by Edisoft.
- Very well adapted to the PLATO needs with a sharing between cores minimizing the interferences.
- Compatible with the LESIA technical heritage (GERICOS platform)
- More predictable and simpler execution model:
  - Less inter-process communication and contention for shared resources
  - Easier to reason about the schedulability of individual tasks



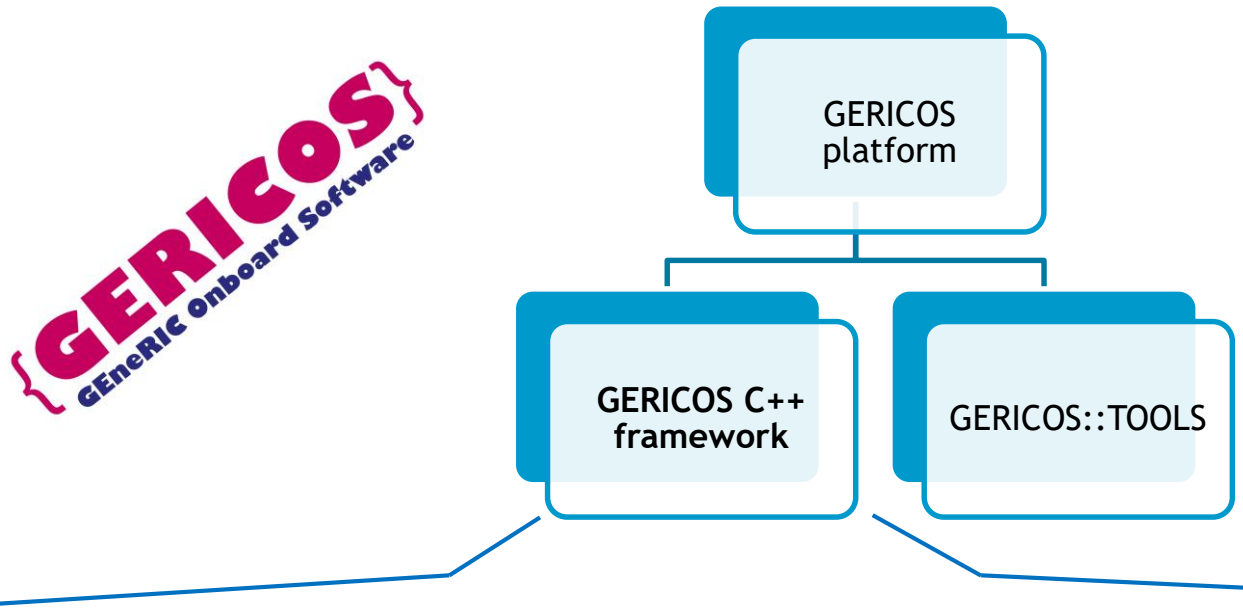
Contents

PLATO N-DPU ASW dual-core architecture and the V&V approach followed

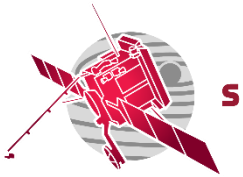
AMP approach: the main issues to solve

- AMP approach poses various technical implementation problems, as well as software engineering problems.
  - How do we make the applications running on each core communicate and collaborate effectively?
  - How do we manage the allocation of shared hardware resources between applications?
  - How do we describe the architecture of these applications?
  - How do we unify the construction of the different applications that run on each core into a single project?
  - How do we analyse the real-time scheduling of these applications and measure the final system performance?
- In the context of the PLATO N-DPU ASW development, a technical answer has been provided to all those questions by enriching the GERICOS platform.

GERICOS platform overview: the C++ framweork

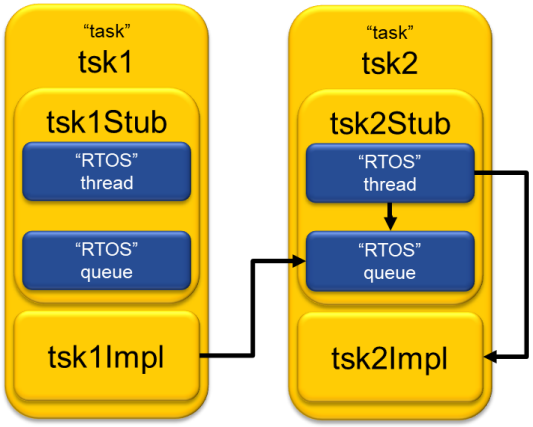


Used in flight on the RPW instrument



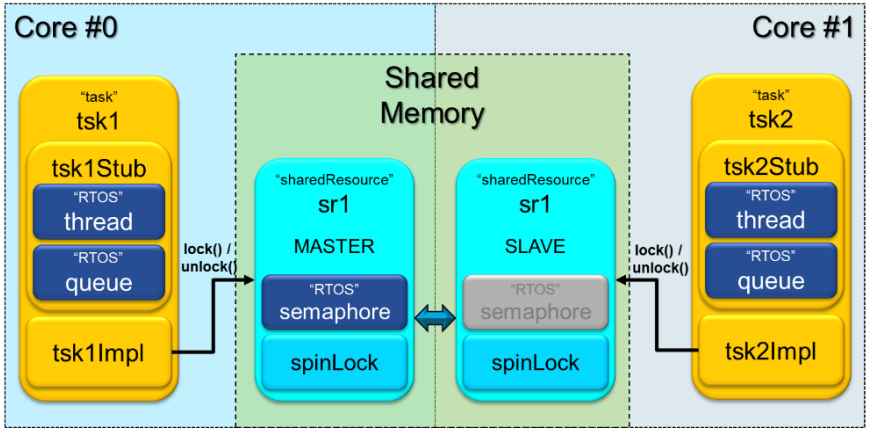
solar orbiter

- GERICOS C++ framework = lightweight, optimized and space qualified C++ implementation of the active object paradigm on top of a real-time kernel (e.g. RTEMS by Edisoft)
  - A real-time application is built as a set of active objects.
  - Each active object (a "task") has its own message queue and computational thread.
- Other concepts included in the GERICOS::CORE middleware:
  - Synchronized objects, shared resources
  - Circular buffers and FIFO
  - Interrupt handlers
- GERICOS::BLOCKS: PUS, data pool, TC/TM, modes, etc.

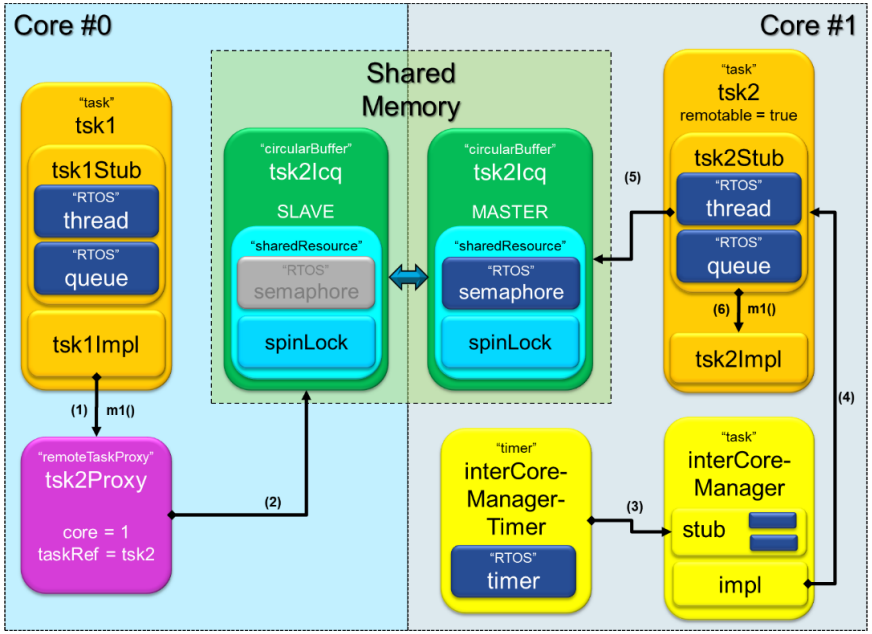


Support of AMP in GERICOS C++ framework

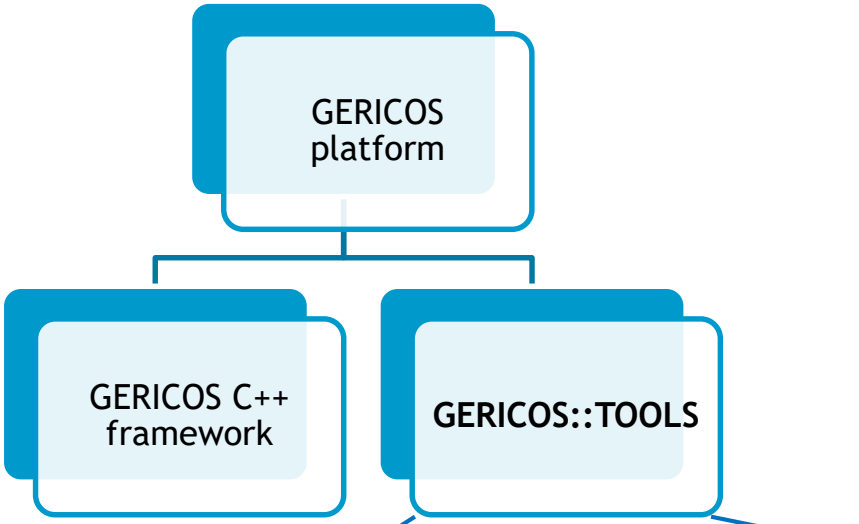
- Resources shared by several cores
  - A new spin lock component has been added to the GERICOS framework.
  - With GERICOS, the shared resources are defined thanks to a specific component encapsulating a RTOS mutex and offering **lock** and **unlock** operations.
  - This GERICOS **shared object** component has been extended with a spin lock so that the resources, like FIFO, can be shared between cores.



- Inter-core task communication
  - A task, running on a first core, can send a message to a remote task, running on a second core, through the use of a proxy and inter-core queues.
  - On the second core, a system task is responsible for polling all inter-core queues and posting the messages retrieved to the corresponding task queues.

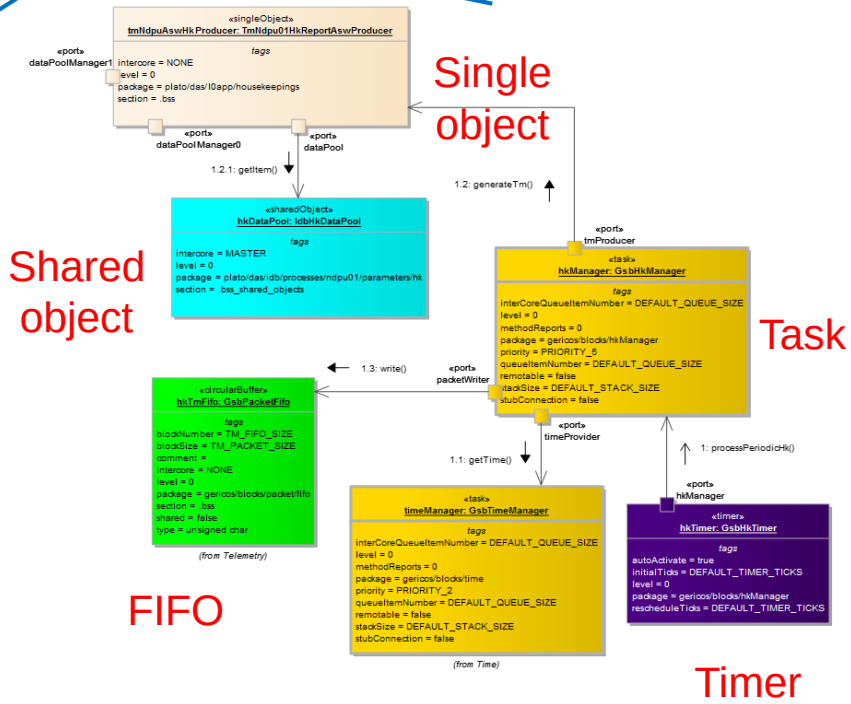


GERICOS platform overview: the toolbox



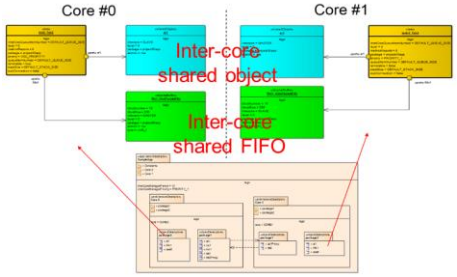
Set of tools for automatizing the development process of embedded S/W:

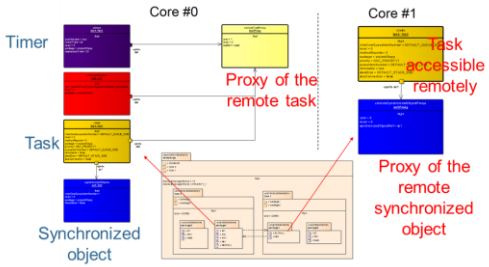
- GERICOS UML profile to describe the static architecture of an embedded application.
- C++ code generation
- Building chain
- …

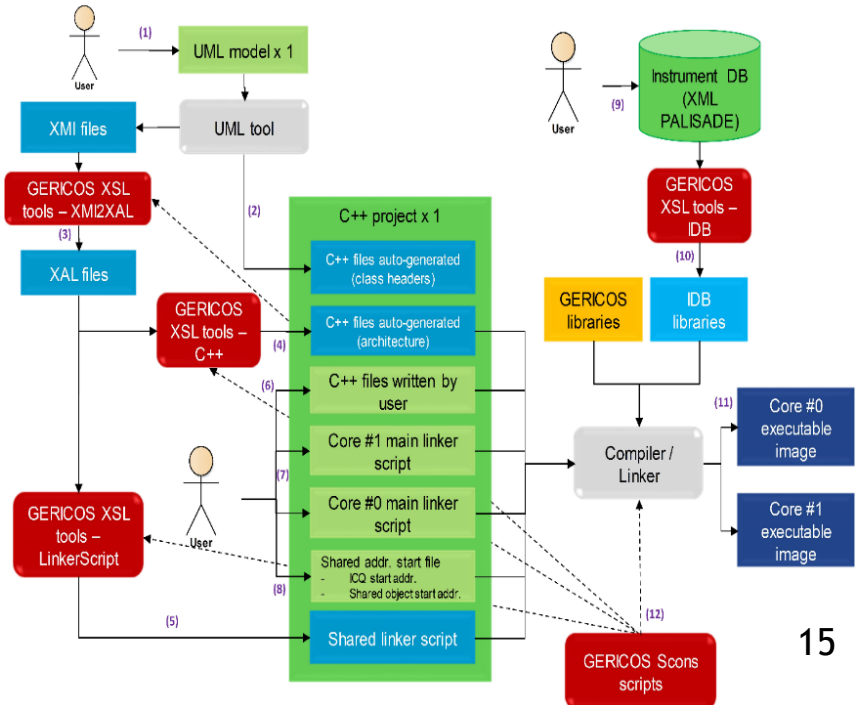


Support of AMP in GERICOS::TOOLS

- GERICOS UML profile
  - The UML profile has been extended to allow for the development of multi-core applications, including the declaration of inter-core shared objects and remote tasks.
- C++ code generation
  - GERICOS::TOOLS can generate C++ code for inter-core system tasks and proxy objects representing locally a remote task.
- Automated management of memory partitioning
  - GERICOS::TOOLS automates the allocation of inter-core shared memory, making it easier to develop applications for AMP dual-core architectures.
- AMP application building automation
  - GERICOS::TOOLS provides an automated software building process for AMP dual-core architectures.
  - This process automatically generates two consistent executable images from one UML model and one C++ project.







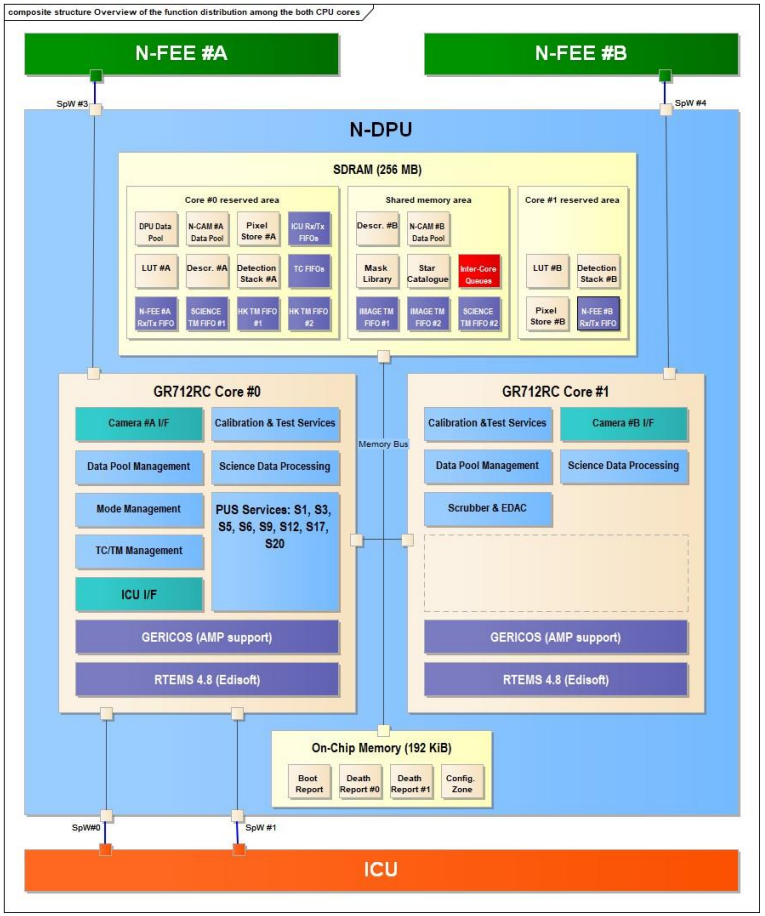
Contents

PLATO N-DPU ASW dual-core architecture and the V&V approach followed

Verification and validation approach

- The N-DPU ASW has a real-time architecture made complex by the number of real-time components but also by the interactions between the 2 cores.
- A failure to conform to timing constraints can result in a loss of science data and a degradation of the instrument performance.

Real time components	CPU core#0 App	CPU core#1 App
Interrupts	3	1
Tasks	20 (1 inter-core)	8 (4 inter-core)
Timers	18	8
FIFO	15 (3 inter-core)	6 (3 inter-core)
Shared objects	11 (6 inter-core)	6 (6 inter-core)
Synchronization objects	3 (1 inter-core)	2



Verification and validation approach

- What are the big challenges to verify and validate such a dual-core software?
- Of all the activities involved in the validation and verification, those most impacted by the dual-core AMP architecture are those linked to technical budgeting and, in particular, **scheduling analysis**.
- The other activities like the tests or the code robustness analysis, are clearly less impacted or not impacted at all.

Activities	Impacts due to AMP dual core architecture?
Unit tests	No (95% of the source code of core#1 app is also used by core#0 app)
Integration tests	Few (additional tests required for the inter-core mechanisms)
Validation tests	No (black box testing, where the two applications are considered as a single system)
Code reliability and robustness analysis	No specific issues brought by the AMP architecture
Coding rules and metrics	No specific issues brought by the AMP architecture
Technical budgets including **scheduling analysis**	**Strong. Scheduling analysis tricky to obtain because of the multi-core architecture**

Schedulability verification

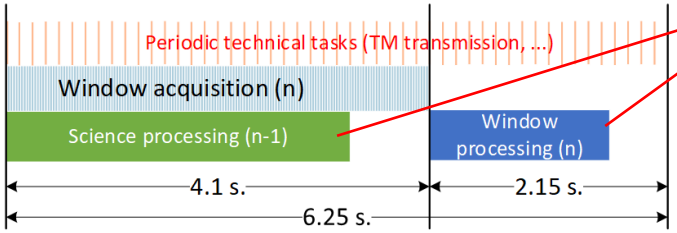
- An **AADL** (Analysis and Design Architecture Language) model of the PLATO N-DPU ASW has been designed with the collaboration of CNES to describe all its dynamic properties:
  - task periods
  - WCETs and deadlines
  - shared resource accesses
  - synchronization mechanisms
  - task precedence constraints
  - …

```
THREAD INTERCOREMANAGER_0_processInterCoreQueues_thr
FEATURES
    -- eventProducerProxy is on CORE 1, so CORE 0 directly access the ICQ, only protected by Spinlock
    eventProducerProxyIcqSl_dat : REQUIRES DATA ACCESS ndpu_data_pkg::eventProducerProxyIcqSl_dat.impl;

PROPERTIES
    POSIX_Scheduling_Policy => SCHED_FIFO; --NO TIMESLICE --TODO: SEE HOW WE CAN ADD LIFO POLICY
    Dispatch_Protocol => Periodic;
    Priority => 254; --HIGHEST PRIORITY: 255, LOWEST PRIORITY: 1 (RTEMS INVERTED)
    Period => 50ms;
    Deadline => 45ms;
    Compute_Execution_Time => 1ms..1ms;
    First_Dispatch_Time => 0ms;
END INTERCOREMANAGER_0_processInterCoreQueues_thr;

THREAD IMPLEMENTATION INTERCOREMANAGER_0_processInterCoreQueues_thr.impl
END INTERCOREMANAGER_0_processInterCoreQueues_thr.impl;
```

- The goal was to propose a model using simplification hypothesis
  - as little pessimistic as possible
  - sufficiently representative to guarantee the schedulability analysis validity
- Modelling hypothesis have been proposed for caches, DMA burst transfers and memory access slowdowns.
- The Cheddar Scheduling Analysis tool fed with the AADL model has been used to obtain the schedulability proof.

Schedulability verification

- At SW-PDR level (2019), long before the scientific algorithms were implemented, the model has proved, by static analysis and simulation, that all the tasks, and in particular the most critical ones, theoretically cannot miss their deadlines.

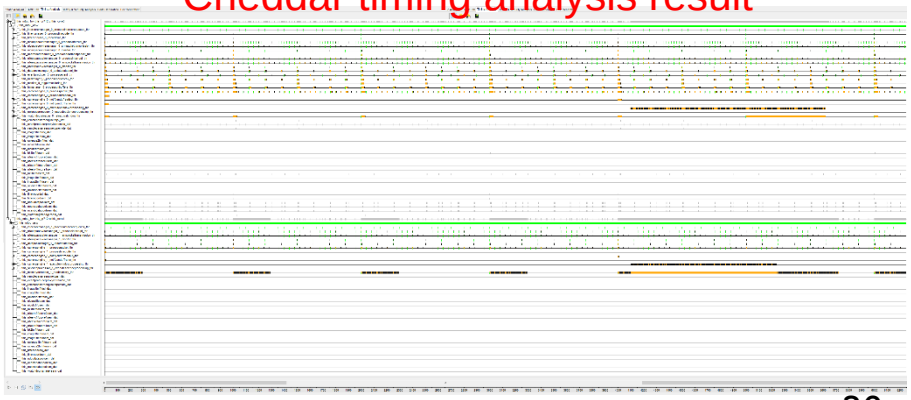


Static Analysis | LAMP Lab | Timing Analysis | Safety & Security Analysis | Code Generation | Doc Generation

	Deadline	Computed	Max Cheddar	Max Marzhin	Avg Cheddar	Avg Marzhin	Min Cheddar
this_ndpu_hw.this_gr712rc.this_core0		0.00 %		0.00 %			
this_ndpu_asw							
this_intercoremanager_0_processintercorequeues_thr	45		1		1.00		1
this_timemanager_0_processtimecode_thr	10		2		2.00		2
this_timerecorder_0_recordtime_thr	700		3		3.00		3
this_nicuspacewiremanager_0_processinterrupt_thr	18		4		1.24		1
this_nicuspacewiremanager_0_armpacketransmission_thr	36		5		2.28		2
this_nicuspacewiremanager_0_monitor_thr	113		6		2.13		1
this_heartbeatproducer_0_produceheartbeatpacket_thr	100		8		8.00		8
this_nfeespacewiremanager_0_processinterrupt_thr	36		7		3.32		3
this_nfeespacewiremanager_0_armpacketransmission_thr	36		9		4.36		4
this_nfeespacewiremanager_0_monitor_thr	113		10		3.51		2
this_datapoolmanager_0_updatedatapool_thr	113		16		12.38		10
this_eventproducer_0_processevent_thr	90		18		7.51		4
this_hkmanager_0_processperiodichk_thr	113		23		15.38		12
this_monitor_0_triggermonitoring_thr	113		24		20.53		17
this_tmmanager_0_processtmbuffers_thr	113		31		23.76		20
this_cameraengine_0_processpacket_thr	36		33		7.85		5
this_cameraengine_0_processtimecode_thr	36		34		28.44		22
this_cameraengine_0_notifyendofreadout_thr	100		33		28.33		22
this_cameraengine_0_notifyendofframe_thr	90		34		29.11		23
this_cameraengine_0_executewindowprocessing_thr	2150		1552		1529.56		1521
this_scienceprocessor_0_executescienceprocessing_thr	4100		3616		3610.25		3605
this_watchdogreloader_0_reloadwatchdog_thr	6250		3617		1160.56		33
this_ndpu_hw.this_gr712rc.this_core1		0.00 %		0.00 %			
this_ndpu_asw							
this_intercoremanager_1_processintercorequeues_thr	45		1		1.00		1
this_nfeespacewiremanager_1_processinterrupt_thr	36		2		1.20		1
this_nfeespacewiremanager_1_armpacketransmission_thr	36		3		2.20		2
this_nfeespacewiremanager_1_monitor_thr	113		4		1.75		1
this_datapoolmanager_1_updatedatapool_thr	113		7		4.75		4
this_cameraengine_1_processpacket_thr	36		9		3.36		3
this_cameraengine_1_processtimecode_thr	36		10		7.11		6
this_cameraengine_1_notifyendofreadout_thr	100		9		7.00		6
this_cameraengine_1_notifyendofframe_thr	90		10		7.78		7
this_cameraengine_1_executewindowprocessing_thr	2150		1142		1140.33		1139
this_scienceprocessor_1_executescienceprocessing_thr	4100		2699		2697.50		2696
this_memoryscrubber_1_readmemory_thr	6250		2986		1290.11		294

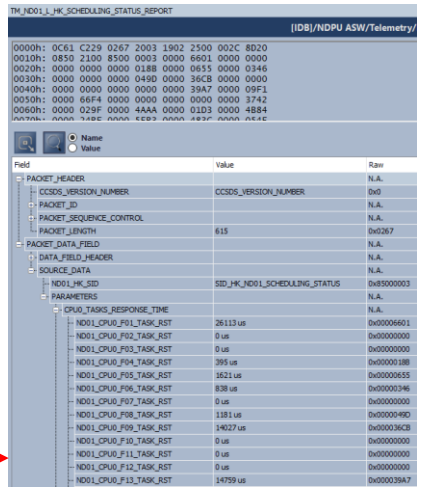
Cheddar timing analysis result

- Moreover, we were able to compute CPU margins from the model.
- Since the model was significantly pessimistic, due to hypothesis we took, we were confident that real CPU margins are above the one computed from the model.



Schedulability verification

- At SW-CDR level (2022) and for the ASW V1 delivery (2023), the schedulability proof was confirmed by direct measurements reported by the flight software itself in its housekeeping parameters
  - thanks to a GERICOS feature allowing to record in real-time the Worst Case Response Times (WRT) of the various tasks
  - using a worst case test scenario whose duration is significant



WRT of each task reported in HK TM packets

- The WRT of each task is compared to the deadlines to check that no violation occurs and to analyse the margins.

Core	Task priority	Task	Threaded function	Response time (µs)	Deadline (µs)	% (Deadline-WRT)	Call count
0	PRIORITY_01	interCoreManager	processInterCoreQueues()	26113	50000	52,2%	39616
0	PRIORITY_05	timeManager	processTimecode()	395	10000	4,0%	1981
0	PRIORITY_07	timeRecorder	recordTime()	1621	700000	0,2%	317
0	PRIORITY_08	nicuSpacewireManager	processInterrupt()	838	30000	2,8%	13997
0	PRIORITY_08	nicuSpacewireManager	armPacketTransmission()	1181	40000	3,0%	12011
0	PRIORITY_08	nicuSpacewireManager	monitor()	14027	125000	11,2%	15847
0	PRIORITY_12	heartbeatProducer	produceHeartbeatPacket()	2545	100000	2,5%	1981
0	PRIORITY_13	nfeeSpacewireManager	processInterrupt()	26356	40000	65,9%	27721
0	PRIORITY_13	nfeeSpacewireManager	armPacketTransmission()	0	40000	0,0%	0
0	PRIORITY_13	nfeeSpacewireManager	monitor()	14146	125000	11,3%	15847
0	PRIORITY_15	highTcReceiver	processPacket()	671	40000	1,7%	5
0	PRIORITY_16	dataPoolManager	updateDataPool()	19114	125000	15,3%	15847
0	PRIORITY_17	hkManager	processPeriodicHk()	19332	125000	15,5%	15846
0	PRIORITY_18	eventProducer	process{$EventName}()	467	100000	0,5%	634
0	PRIORITY_19	monitor	triggerMonitoring()	9407	500000	1,9%	3962
0	PRIORITY_20	tmManager	processTmBuffers()	24243	125000	19,4%	15846
0	PRIORITY_22	cameraEngine	processPacket()	18492	68000	27,2%	27404
0	PRIORITY_22	cameraEngine	processTimecode()	1358	40000	3,4%	317
0	PRIORITY_22	cameraEngine	notifyEndOfReadout()	363	100000	0,4%	317
0	PRIORITY_22	cameraEngine	notifyEndOfFrame()	239	100000	0,2%	317
0	PRIORITY_22	cameraEngine	executeWindowProcessing()	1198783	2150000	55,8%	317
0	PRIORITY_24	medTcReceiver	processPacket()	2673	2150000	0,1%	5
0	PRIORITY_25	scienceProcessor	executeScienceProcessing()	3529192	5051217	69,9%	317
0	PRIORITY_26	windowProgrammer	executeWindowProgrammation()	0	75000000	0,0%	0
0	PRIORITY_27	lowTcReceiver	processPacket()	1854	75000000	0,0%	2
0	PRIORITY_29	watchdogReloader	reloadWatchdog()	4494077	75000000	6,0%	1981
1	PRIORITY_01	interCoreManager	processInterCoreQueues()	25100	50000	50,2%	39615
1	PRIORITY_04	nfeeSpacewireManager	processInterrupt()	25728	40000	64,3%	27722
1	PRIORITY_04	nfeeSpacewireManager	armPacketTransmission()	0	40000	0,0%	0
1	PRIORITY_04	nfeeSpacewireManager	monitor()	7358	125000	5,9%	15847
1	PRIORITY_06	dataPoolManager	updateDataPool()	9725	125000	7,8%	15847
1	PRIORITY_08	cameraEngine	processPacket()	3213	68000	4,7%	27405
1	PRIORITY_08	cameraEngine	processTimecode()	796	40000	2,0%	317
1	PRIORITY_08	cameraEngine	notifyEndOfReadout()	423	100000	0,4%	317
1	PRIORITY_08	cameraEngine	notifyEndOfFrame()	248	100000	0,2%	317
1	PRIORITY_08	cameraEngine	executeWindowProcessing()	1146509	2150000	53,3%	317
1	PRIORITY_10	scienceProcessor	executeScienceProcessing()	3238686	5103491	63,5%	317
1	PRIORITY_12	windowProgrammer	executeWindowProgrammation()	0	75000000	0,0%	0
1	PRIORITY_30	memoryScrubber	readMemory()	4413639	75000000	5,9%	1979

Thank you / Questions ?

philippe.plasson@obspm.fr