www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

ePAPI: Performance Application Programming Interface for embedded platforms

Jeremy Giesen[§¶], Enrico Mezzetti[§], Jaume Abella[§],
Enrique Fernández[†], Francisco J. Cazorla[§]

§ Barcelona Supercomputing Center
Centro Nacional de Supercomputación

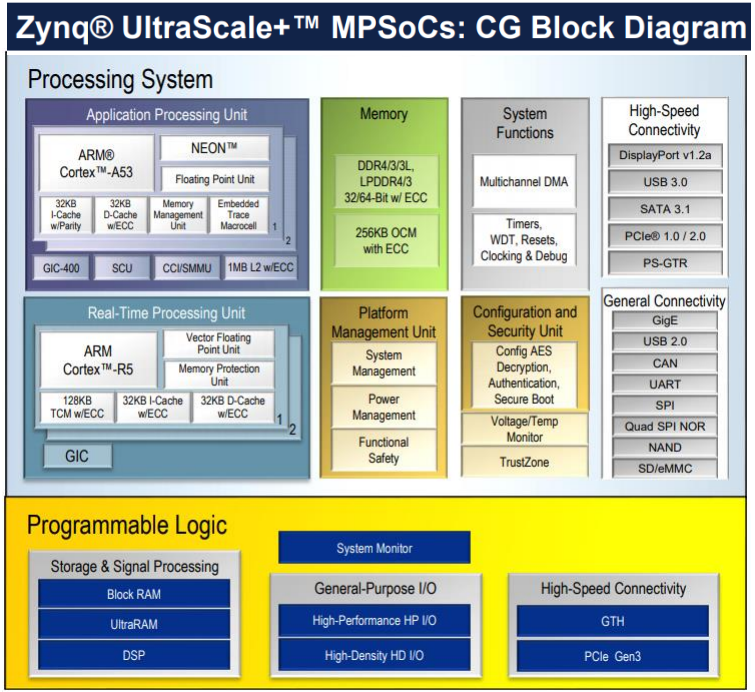
¶ UNIVERSITAT POLITÈCNICA DE CATALUNYA

† UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

19th International Workshop on Worst-Case Execution Time Analysis - July 9, 2019

COTS Multicore processors

- **Heavily used in critical systems**
  - Even the most conservative system domains
- **Features**
  - Multiple level caches, clusters of cores, hardware accelerators, FPGAs, …
- **Pros**
  - Cover the increasingly computational requirements of advanced software functionalities
    - E.g. Autonomous Driving, in automotive
- **Cons**
  - They hamper the effectiveness of consolidated WCET analysis approaches



Event Monitors: At the Heart of Predictability

- **HW support has been proposed to improve predictability**
  - Those solutions have not been fully adopted by industry yet for cost reasons
  - Yet some techniques have hit the silicon: cache partitioning
- **SW support for chips with limited HW support for predictability**
  - Goal:
    - Control contention among tasks to different shared resources.
  - How? Via the operating system/hypervisor
    - Monitors task's activities using the available PMCs and
    - Suspends or restrains tasks' execution when their assigned budget is exhausted.

Event Monitors: At the Heart of Predictability

- **HW support has been proposed to improve predictability**
  - Those solutions have not been fully adopted by industry yet for cost reasons

Event monitors,
and the SW-visible counter to access them (PMCs),
are at the heart of SW solutions for time predictability

  - How? Via the operating system/hypervisor
    - Monitors task's activities using the available PMCs and
    - Suspends or restrains tasks' execution when their assigned budget is exhausted.

Mastering Diversity

- **Structured approach to use PMCs for platform analysis**
  - Helps mastering complex PMUs (hundreds of event monitors)
    - abstract away from the low-level hardware details
    - guarantee a correct manipulation of the registers
  - Masters diverse PMC support inter- and intra-platform
- **In this respect, a standardization effort in the high-performance and mainstream domains to configure and use monitoring counters**
  - Kernel-level tools (perf)
  - Shared common libraries
    - Performance Application Programming Interface (PAPI)

Mastering Diversity

- **Structured approach to use PMCs for platform analysis**
  - Helps mastering complex PMUs (hundreds of event monitors)
    - abstract away from the low-level hardware details

No equivalent solution is currently available for embedded reference platforms and RTOSes

configure and use monitoring counters

  - kernel-level tools (perf)
  - Shared common libraries
    - Performance Application Programming Interface (PAPI)

Our work

- **First step towards filling this gap**
  - Address the implementation of an abstract PMC library to collect in a platform-independent way relevant events
  - To which extent PAPI could be used for fine-grained platform time analysis?
- **More specifically:**
  - Assess PAPI compatibility with PMC support on AURIX ™ TC297
    - Reference platform in the automotive domain
  - Define, implement and validate ePAPI a functionally-equivalent, low-overhead port of PAPI to the referenced platform

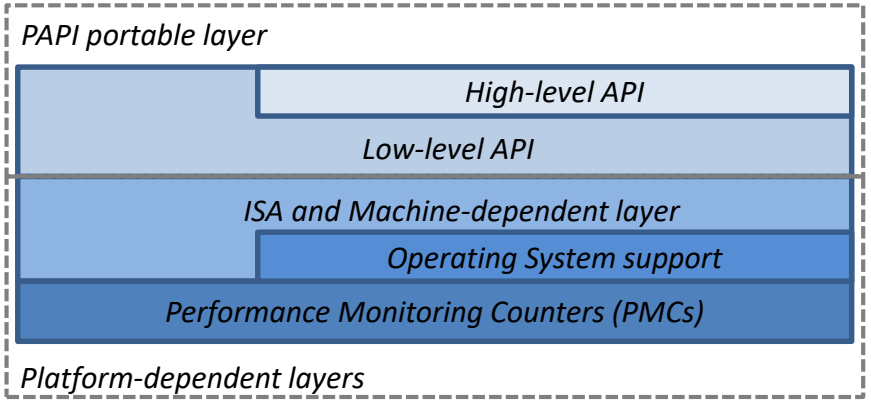
Outline

- Motivation
- **Introduction to PAPI**
- **Porting PAPI to an embedded platform**
  - The reference platform with emphasis on event monitors
  - Selection and mapping of PAPI events
- **ePAPI**
  - Implementation
  - Validation
- **Conclusions and Future Work**

Introduction to PAPI

Performance Application Programming Interface

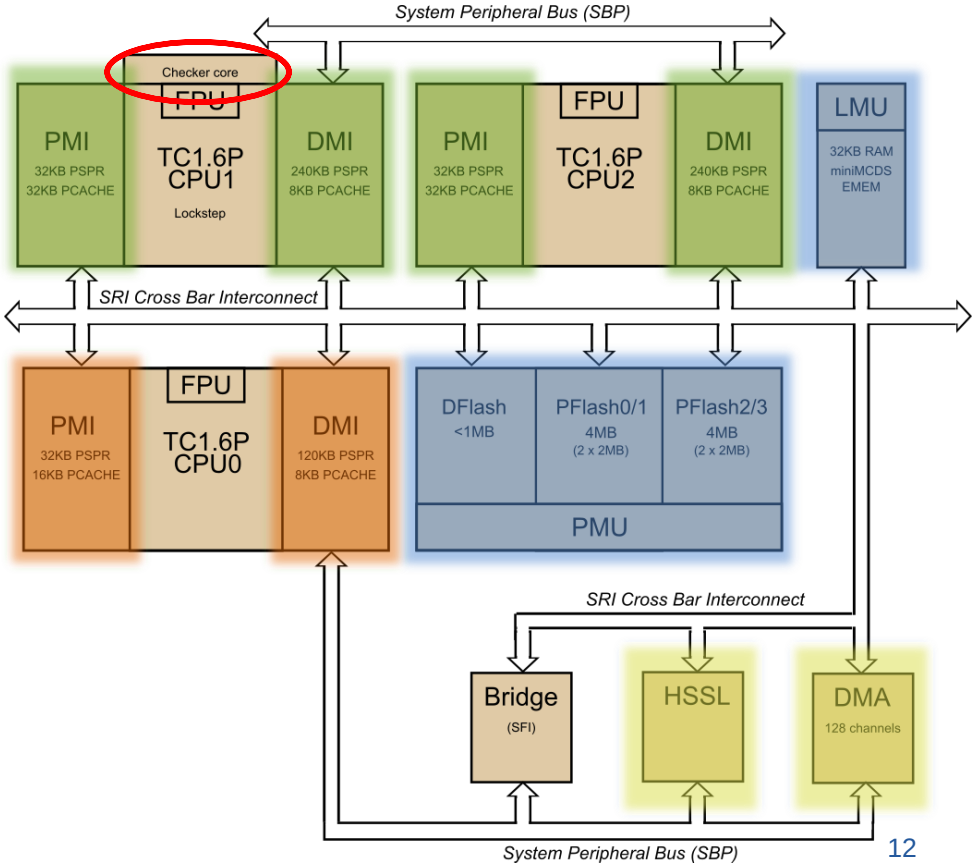
- **Cross-platform library with supporting utilities for application profiling**
  - De-facto standard for mainstream and HPC COTS hardware platform
  - Monitors collect statistic on the occurrence of specific hardware events
  - Two-layered approach enables sharing the same interface across platforms
- **User-level standardized API**
  - Unified interface across families of processors
  - For coarse-grained and fine-grained profiling
- **Machine-dependent part**
  - Enables configuration and collection of PMC data
  - ISA and platform specific
  - Partial interface compliance is admissible



Porting PAPI to an embedded platform

Infineon AURIX TC 297 ED

- **TriCore TC1.6 P**
  - Private instr and data caches
  - Private instr and data scratchpads
  - Superscalar
    - Integer, Load/Store and Loop pipelines
  - Not exactly the same config though
    - One core operates in lockstep mode
    - One cores equips smaller local memories
- **SRI Cross Bar Interconnect**
  - Shared PMU (Program Memory Unit)
    - DFlash and 4 PFlash SRI interfaces
  - Shared LMU (Local Memory Unit)
    - Relatively small shared RAM
- **System Peripheral Bus**
  - Supports DMA and HSSL transactions



TC 297 Debug Support

- **Each TC1.6 P features a relatively small PMU (Perfomance Monitoring Unit)**
  - 5 PMC registers per core
  - Two dedicated registers for cycle (CCNT) and instruction (ICNT) counters
  - Three multiplexed registers (M1CNT, M2CNT, and M3CNT)
    - Multiplexed PMCs can be configured to track one of the hardware events supported by the PMU

Event	Description
IP_DISPATCH_STALL	Incremented each cycle the Integer dispatch unit is stalled for whatever reason.
LS_DISPATCH_STALL	Incremented each cycle the Load-Store dispatch (LSU) unit is stalled for whatever reason.
LP_DISPATCH_STALL	Incremented each cycle the Loop dispatch unit is stalled for whatever reason.
MULTI_ISSUE	Incremented each cycle when more than one instruction is issued.
PCACHE_HIT	Incremented each time the fetch unit is (NOT) found in the program cache.
PCACHE_MISS	
DCACHE_HIT	Incremented each time the target of a cached request from the Load-Store unit is found in the data cache.
DCACHE_MISS_CLEAN	Incremented each time the target of a cached request from the Load-Store unit is not found in the data cache & hence a bus fetch is initiated with no dirty cache line eviction/write-back of a dirty line.
DCACHE_MISS_DIRTY	
TOTAL_BRANCH	Incremented each cycle a branch instruction is in a branch resolution stage of the pipeline.
PMEM_STALL	Incremented each cycle the FU is requesting an instruction and the imem is stalled
DMEM_STALL	Incremented each cycle the LSU is requesting a data operation and the dmem is stalled

TC 297 Debug Support

- **Supported hardware events**
  - Only 12 hardware events can be tracked through the multiplexed counters
  - The Counter Control Register (CCTRL) can be configured to select the event to be tracked
    - The CCTRL register is configured via assembly language
    - Only 3 events at a time (summing up to the 2 static CCNT and ICNT)
  - Not all combinations are allowed

CCTRL bits	M1CNT	M2CNT	M3CNT
000	IP_DISPATCH_STALL	LS_DISPATCH_STALL	LP_DISPATCH_STALL
001	PCACHE_HIT	PCACHE_MISS	MULTI_ISSUE
010	DCACHE_HIT	DCACHE_MISS_CLEAN	DCACHE_MISS_DIRTY
011	TOTAL_BRANCH	PMEM_STALL	DMEM_STALL

Selection and mapping of PAPI events

- **PAPI specification includes 100+ preset events**
  - However, actual implementations typically support a subset
- **Porting to the TC 297 is not an exception**
  - Events inherently unsupported
    - E.g., 30+ events related to L2 and L3 caches
  - Events voluntarily discarded
    - We focused on events we considered relevant for the embedded domain (timing and energy, multicore contention, average performance analyses)
    - PAPI_RES_STL – Cycles stalled on any resource.
- **Supported events**
  - 15 events of which 9 are PAPI preset events and 6 are native events
  - Event mapping straightforward except for some cases
    - Some preset events can be mapped to the combination of more than one PMC
    - In few cases the TC 297 PMU could only provide an over-approximation of the PAPI event

ePAPI Implementation and Validation

ePAPI Implementation

- **Full PAPI specification includes 70+ functions**
  - We restricted to functions in reason of the hardware-software configuration (TC 297, bare-metal setting)
- **Current ePAPI support**
  - Almost all (7 out of 10) functions defined by PAPI high-level interface
    - Related to component selection and high-level FPU statistics
  - A small subset (17 out of 66) of PAPI low-level interface
    - Focus on CPU performance
    - Lack of platform support for Floating-Point Unit (FPU) events
    - No operating system support
    - No standard output

PAPI vs ePAPI signatures equivalence

```
int Events[NUM_EVENTS] = {PAPI_TOT_CYC};
long long values[NUM_EVENTS];

/* Start counting events */
PAPI_start_counters(Events, NUM_EVENTS);

//Place code under analysis here

/* Read counters */
PAPI_read_counters(values, NUM_EVENTS);

//Place more code under analysis here

/* Add the counters */
PAPI_accum_counters(values, NUM_EVENTS);

//Place more code under analysis here

/* Stop counting events */
PAPI_stop_counters(values, NUM_EVENTS);
```

```
int Events[NUM_EVENTS] = {PAPI_TOT_CYC};
long long values[NUM_EVENTS];

/* Start counting events */
ePAPI_start_counters(Events, NUM_EVENTS);

//Place code under analysis here

/* Read counters */
ePAPI_read_counters(values, NUM_EVENTS);

//Place more code under analysis here

/* Add the counters */
ePAPI_accum_counters(values, NUM_EVENTS);

//Place more code under analysis here

/* Stop counting events */
ePAPI_stop_counters(values, NUM_EVENTS);
```

ePAPI validation

- **ePAPI implementation needs to be assessed on**
  - Accuracy of PMCs on the target platform
    - Pre-requisite for verification and validation
  - Equivalence of functional behavior
    - ePAPI functions shall meet the software specification from PAPI documentation
  - Limited overhead of ePAPI functions
    - Library calls are also required to incur small overhead on PMC values
- **Baseline for assessment**
  - We built on top of `PMClib` [6] a low-level, zero-overhead PMC library

[6] E. Díaz et al. “Modelling multicore contention on the AURIX TC27x” DAC 2018

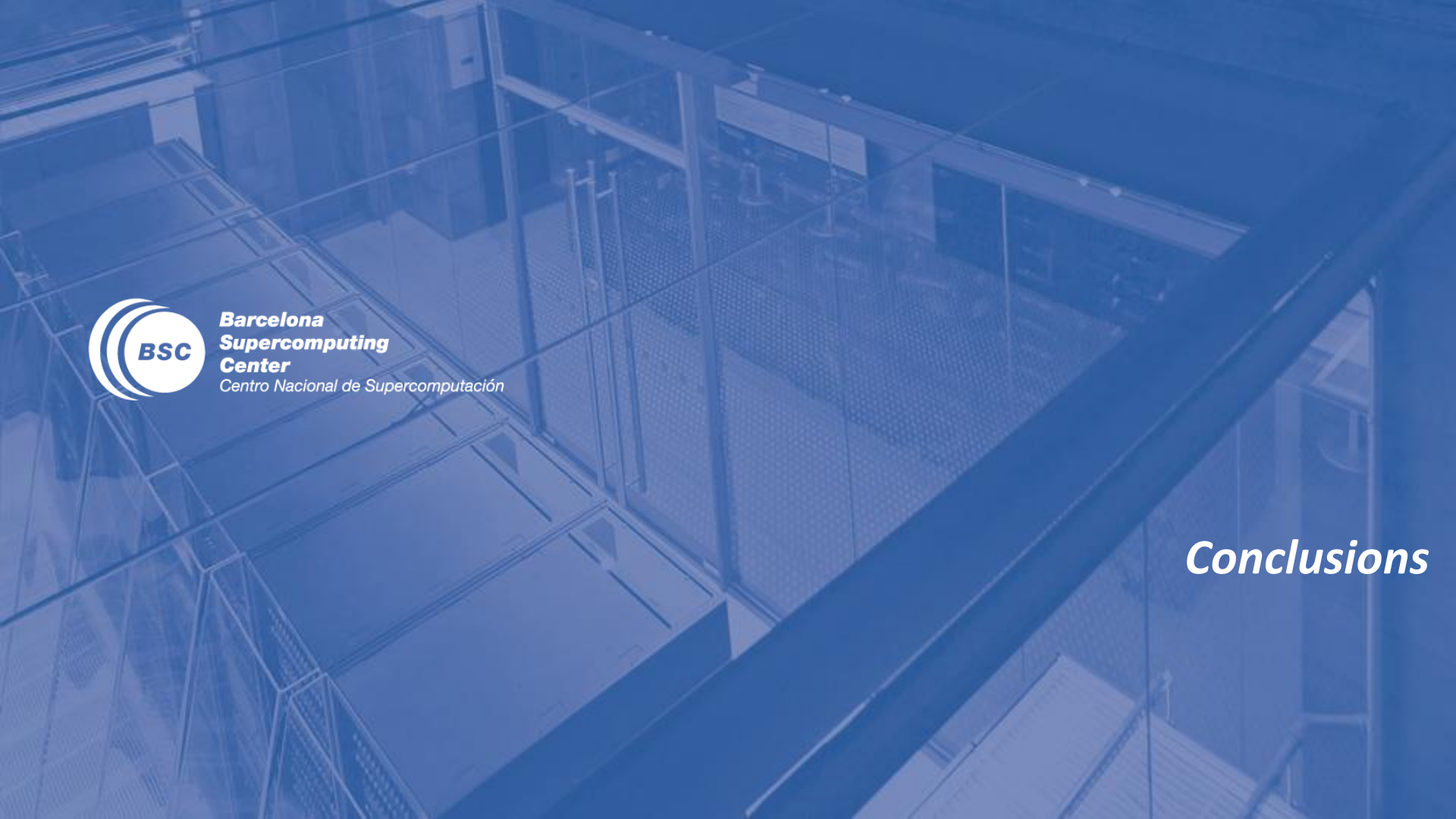
ePAPI validation

- **Accuracy of PMCs**
  - We exploited small ad-hoc benchmarks that cause a known amount of events to be triggered for each traceable event
- **Equivalence of functional behavior**
  - Functional testing campaign on both high- and low-level functions
- **Limited overhead of ePAPI functions**
  - Generic infrastructure of PAPI (ePAPI) high-level interface cannot have zero impact on PMCs
    - It necessarily requires more instructions to be executed
  - Impact must be reasonably low and more importantly bounded

PMClib vs ePAPI plain and accumulated PMC reads

PAPI Event	Bare-metal	ePAPI Read	ePAPI accum.	Bare-metal	ePAPI Read	ePAPI accum.	Bare-metal	ePAPI Read	ePAPI accum.
	Event count			Tot. Cycles			Tot. Instructions		
ePAPI_BRU_IDL	70.017	+27	+28	100.019	+30	+31	80.011	+11	+11
ePAPI_L1_ICA	499.002	+0	+0	1.029.597	+3	+3	1.017.010	+12	+12
ePAPI_L1_ICR	499.002	+0	+0	1.029.597	+23	+23	1.017.010	+12	+12
ePAPI_L1_ICH	498.890	+0	+0	1.029.595	+25	+26	1.017.010	+12	+12
ePAPI_L1_ICM	124.987	+0	+0	2.879.511	+23	+23	1.513.009	+11	+11
ePAPI_L1_DCA	1.000.000	+0	+0	1.029.599	+20	+31	1.017.009	+11	+11
ePAPI_L1_DCH	999.900	+0	+0	1.029.597	+22	+22	1.017.009	+11	+11
ePAPI_L1_DCM	6.001	+0	+0	299.616	+16	+17	32.010	+12	+12
ePAPI_MEM_SCY	290.336	+0	+0	430.579	+25	+25	65.011	+11	+11
PMEM_STALL	127	+3	+3	160.061	+32	+33	143.011	+11	+11
DMEM_STALL	301.714	+0	+0	401.820	+28	+29	40.011	+11	+11
MULTI_ISSUE	100.014	+3	+1	120.040	+33	+34	230.050	+12	+12
IP_DISPATCH_STALL	1.000	+0	+0	17.014	+33	+34	17.010	+12	+12
LS_DISPATCH_STALL	2.002	+10	+10	17.014	+33	+34	17.010	+12	+12
LP_DISPATCH_STALL	1.001	+0	+0	1.008.032	+25	+26	2.006.014	+11	+11

- **Low overhead in all cases**
  - Very few additional event counts
  - Only observed differences were on the number of instructions and cycles,
    - In most cases no overhead on specific event counts
- **Reduction of probe effects**
  - Counters are enabled and disabled to guarantee that the measured events belong to the program under analysis
- **Results are constant**
  - After running several times the benchmarks, the overheads kept constant
  - Even for a similar platform of the same family, the TC275.



Conclusions

Considerations

- Analyzing PAPI interface and target support
  - Identified some limitations and desirable characteristics
- From the PAPI perspective
  - PAPI Preset events are necessarily generic
    - Mostly designed to collect performance metrics for optimization purposes
  - Relevant events from the embedded domain perspective are only supported as native events
  - Not having those events in the cross-platform interface partially defeats the benefits of having a standardized interface
- From the AURIX™ TC297 perspective
  - We suffered from the limited PMC support available, compared to the support available in conventional processors and more advanced embedded targets
    - This was indeed the main cause for discarding PAPI functionalities from the porting
  - PMC support in the TC297 does not allow to characterize contention effects with satisfactory precision as stall events cannot be associated to the different target in the SRI
    - E.g., PMEM_STALL event is counting stalls cycles suffered when fetching code from any PFlash interface, the LMU or even non-local scratchpads

Conclusions and next steps

- **We investigated on a standardized performance monitoring interface for embedded targets**
  - Considered the general-purpose PAPI specification
  - Assessed it against the available PMC support in the AURIX TC 297
  - Developed and validated ePAPI
    - Functionally-equivalent and low-overhead implementation of PAPI for the TC 297
- **Future directions**
  - Extend ePAPI TC 297 to support to RTOS events
    - Compatible Erika RTOS
  - Port ePAPI to different platform
  - Make the implementation available to the community

www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

ePAPI: Performance Application Programming Interface for embedded platforms

Jeremy Giesen[§¶], Enrico Mezzetti[§], Jaume Abella[§],
Enrique Fernández[†], Francisco J. Cazorla[§]

§ Barcelona Supercomputing Center
Centro Nacional de Supercomputación

¶ UNIVERSITAT POLITÈCNICA DE CATALUNYA

† UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

19th International Workshop on Worst-Case Execution Time Analysis - July 9, 2019