Revisiting Resource Partitioning for Multi-core Chips:
Integration of Shared Resource Partitioning on a Commercial RTOS

21 Apr. 2017

PAK,EUNJI

Senior researcher, ETRI

(Electronics and Telecommunications Research Institute)

pakeunji@etri.re.kr

Agenda

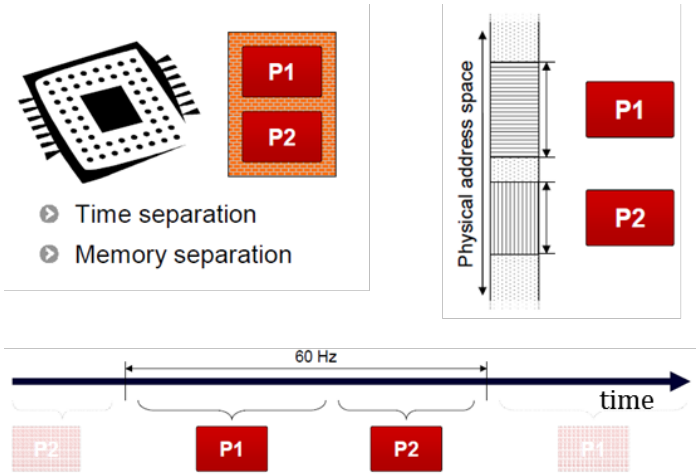
- **Qplus-AIR, a commercial RTOS**
- **Comprehensive shared resource partitioning implementation on Qplus-AIR**

Qplus-AIR

ARINC653 compliant RTOS
Certifiable for DO-178B Level A

Introduction to Qplus-AIR

- **Qplus-AIR**
  - Developed by ETRI for safety-critical system (2010~ 2012)
    - Main operating system for the IFCC (Integrated flight control computer) of UAV (Unmanned Avionics Vehicle), KAI
    - Integrate MC (Mission Control), FC (Flight Control), and C&C (Communications and Commands) in the IFCC
  - ARINC653 compliant RTOS*
  - Robust partitioning among applications
    - Spatial and temporal
    - Prevent cross-application influence and error propagation among applications
    - Easy integration of multiple applications with different degrees of criticality



* Airlines Electronic Engineering Committee, Avionics Application Software Standard Interface ARINC Specification 653 Part 1, 2006

Introduction to Qplus-AIR

- **Qplus-AIR**
  - Certifiable package for DO-178B Level A
  - Lightweight ARINC653 support: kernel-level implementation
  - Support for multicore platforms (2014 ~)
- **RTWORKS**
  - A commercial version of Qplus-AIR
  - Managed by RTST (2013 ~), ETRI's spin-off company
    - Start with 4 developers, and now has 11 OS developers
  - AUTOSAR (automotive industry standard) and ISO 26262 ASIL D is in progress
- **ETRI focuses on research issues while RTST focuses on commercialization**

Application Examples

- **Safety-critical industrial applications**
  - Integrated flight control computer of unmanned avionics vehicle, 2010 ~ 2012
  - Tiltrotor flight control computer, 2012
  - Nuclear powerplant control system, 2013
  - HUMS(Health and Usage Monitoring System) for helicopter, 2013 ~ 2016
  - Subway screen-door control system, 2016 (export to Brazil)
  - Communication system of self-propelled guns, 2017~
  - (project) Autonomous driving car, 2015~

Comprehensive shared resource partitioning implementation on Qplus-AIR

Contents

- **Introduction**
- **HW platform: P4080**
- **Comprehensive resource partitioning implementation**
  - Memory bus bandwidth partitioning
  - DRAM bank partitioning
  - Shared cache partitioning – set-based / way-based
- **Combined all the techniques on the Qplus-AIR**
- **Evaluations**
- **Conclusions & Future Work**

Introduction [1/2]

- **Robust partitioning among applications (partitions)**
  - Qplus-AIR supports spatial and temporal partitioning
  - Ensures independent execution of multiple applications with various safety-critical levels
- **Robust partitioning may no longer be valid in multicore**
  - Multiple cores share hardware resources such as cache or memory
  - Concurrently executing applications affect each other due to the contention on shared resource
  - Major source of timing variability
    - Pessimistic WCET estimation → overprovisioning of hardware resources and low system utilization
  - In safety-critical systems, we had to turn off but one core

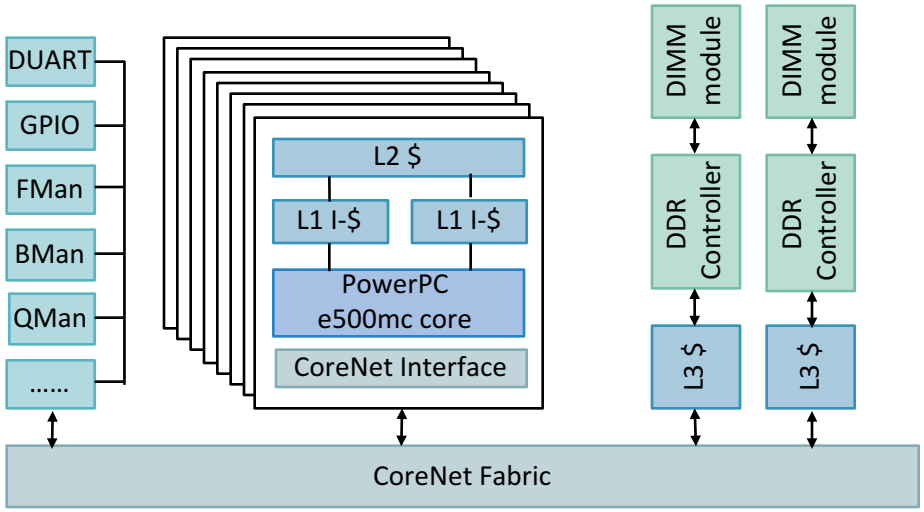
Introduction [2/2]

- **We must deal with the resource contention properly**
  - WCET of tasks stays guaranteed and tightly bounded
  - Especially for safety critical applications that require certification
- **Requirement of inter-core interference mitigation**
  - *"The applicant has identified the interference channels that could permit interference to affect the software applications hosted on the MCP cores, and has verified the applicant's chosen means of mitigation of the interference. "*
    - FAA CAST (Certification Authorities Software Team)-32A Position Paper*
- **Comprehensive shared resource partitioning implementation on ARINC653 compliant RTOS**
  - Integrate a number of resource partitioning schemes, each of which targets different shared hardware resources, on Qplus-AIR
  - Unique challenges due to the fact that the RTOS did not support Linux-like dynamic paging

* Certification Authorities Software Team, Position Paper CAST-32A : Multi-core Processors, 2016.

HW platform, P4080 [1/2]

- **P4080 architecture***
  - Eight PowerPC e500mc cores
  - Each core has a private 32KB-I/32KB-D L1 and 128KB L2 cache
  - Two L3 32-way 1MB caches with cache-line interleaving
  - Two memory controllers for two 2GB DDR DIMM modules (each DIMM modules has 16 DRAM banks)
  - CoreNet coherency fabric – interconnects cores and other SoC modules, a high-bandwidth switch that supports several concurrent transactions

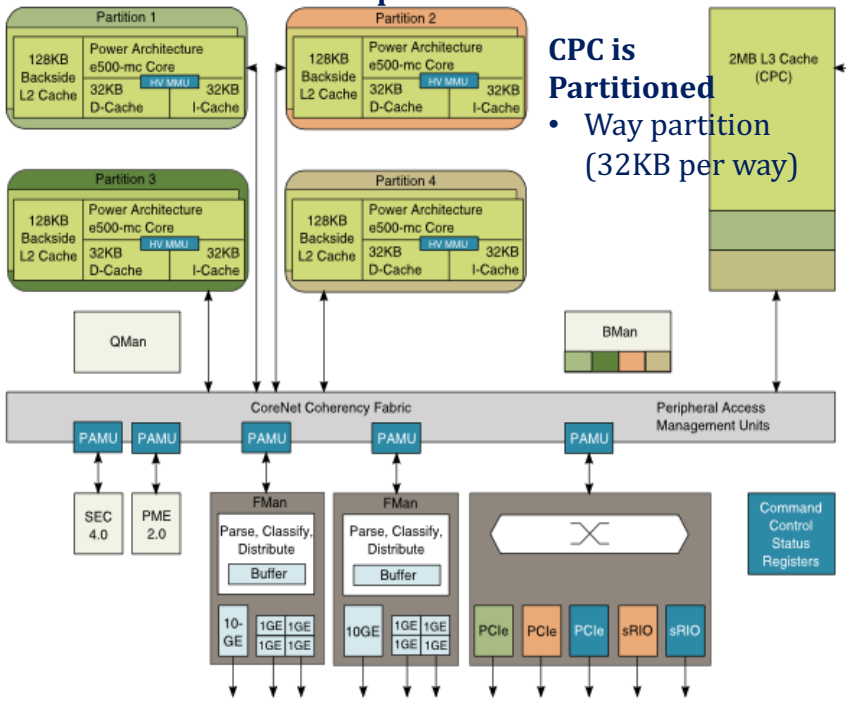


* P4080 QorIQ Integrated Processor Hardware Specifications, Feb 2014.

HW platform, P4080 [2/2]

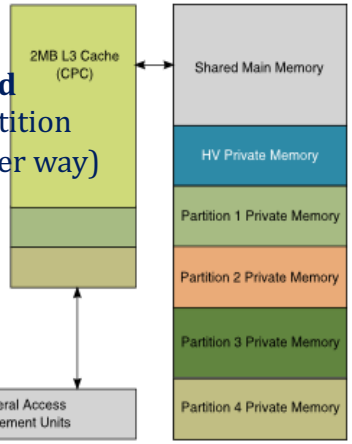
- **Partitioning support of recent PowerPC processors***

Each core is allocated to each partition



CPC is Partitioned

- Way partition (32KB per way)



Main memory is divided into several physical regions

- Private
- Shared between partitions; accessible at user level
- Shared among partitions; restricted to hypervisor level

This mapping is enforced by the core's MMUs

Restrict the coherency overhead

- Disable the coherency – prevent snoop overhead
- Specify a group participating coherency

System peripherals are not shared

- Hypervisor is able to restrict their DMA-accessible memory range to some part of the memory region (through the MMU)

* Hardware Support for Robust Partitioning in Freescale QorIQ Multicore SoCs (P4080 and derivatives)

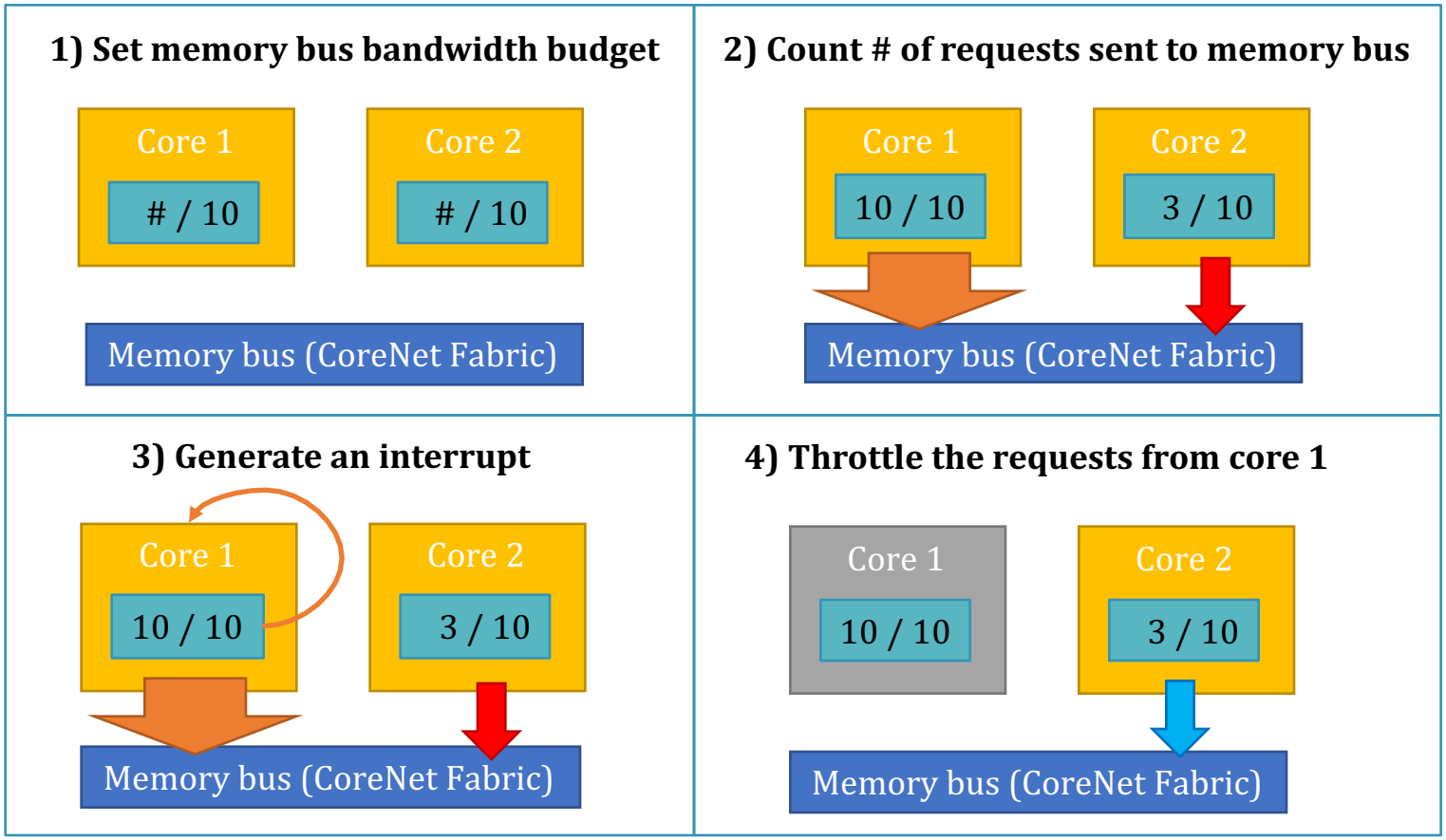
Resource partitioning mechanisms

- **1. Memory bus (interconnect) bandwidth partitioning**
- **2. Memory bank partitioning**
- **Shared cache partitioning**
  - 3. Set-based cache partitioning with page coloring
  - 4. Way-based cache partitioning with the support of P4080 hardware
- **Combine all the techniques and integrated on Qplus-AIR**
- **Paging**
  - Memory bank partitioning and set-based cache partitioning assumes that OS supports Linux-like paging
  - Paging implementation in Qplus-AIR

Resource partitioning mechanisms

Memory bus bandwidth regulator [1/2]

- **Bus bandwidth regulator***
  - Limit the bandwidth usage per core



* H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. Memory bandwidth management for efficient performance isolation in multi-core platforms. IEEE Transactions on Computers, 65:562–576, 2015.

Resource partitioning mechanisms

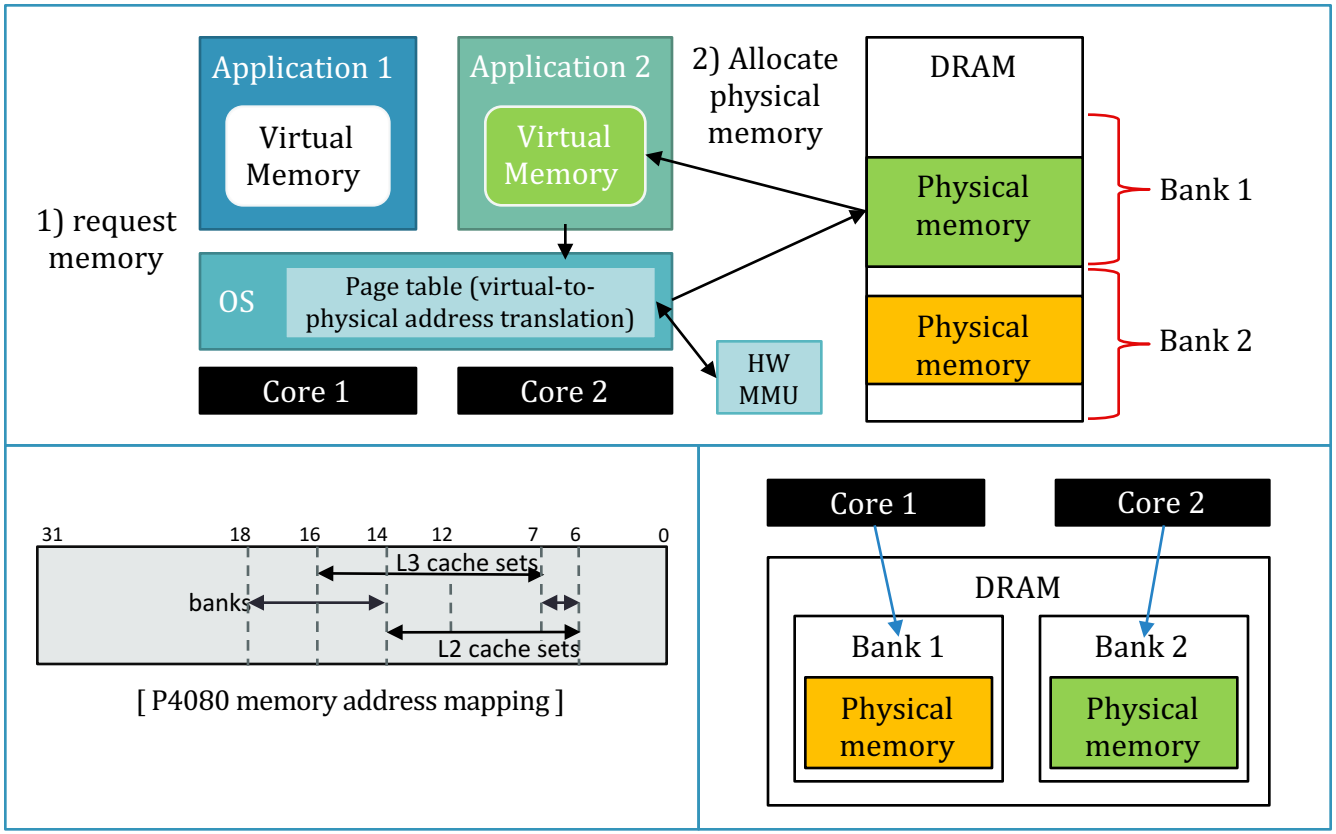
Memory bus bandwidth regulator [2/2]

- **Implementation**
  - Setup the budget and configure to generate an interrupt when a core exhaust the budget
    - Configure performance monitoring control registers and performance monitoring counters
  - OS scheduler throttles further execution at that core
    - Implement interrupt handler for the interrupt that PMC generates
    - Scheduler de-schedule the tasks on the core
- **Period of bandwidth regulator execution**
  - If too short, overhead becomes excessive; in contrast, if too long, predictability is worsened
  - Default period of our implementation is 5ms

Resource partitioning mechanisms

Bank-aware memory allocation

- **DRAM bank-aware memory allocation***
  - Manages memory allocation in such a way that no application shares its memory bank with applications running on other cores

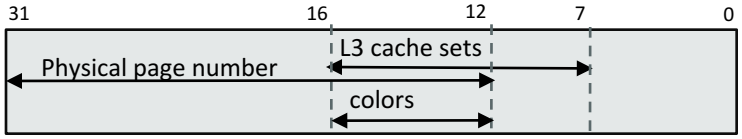


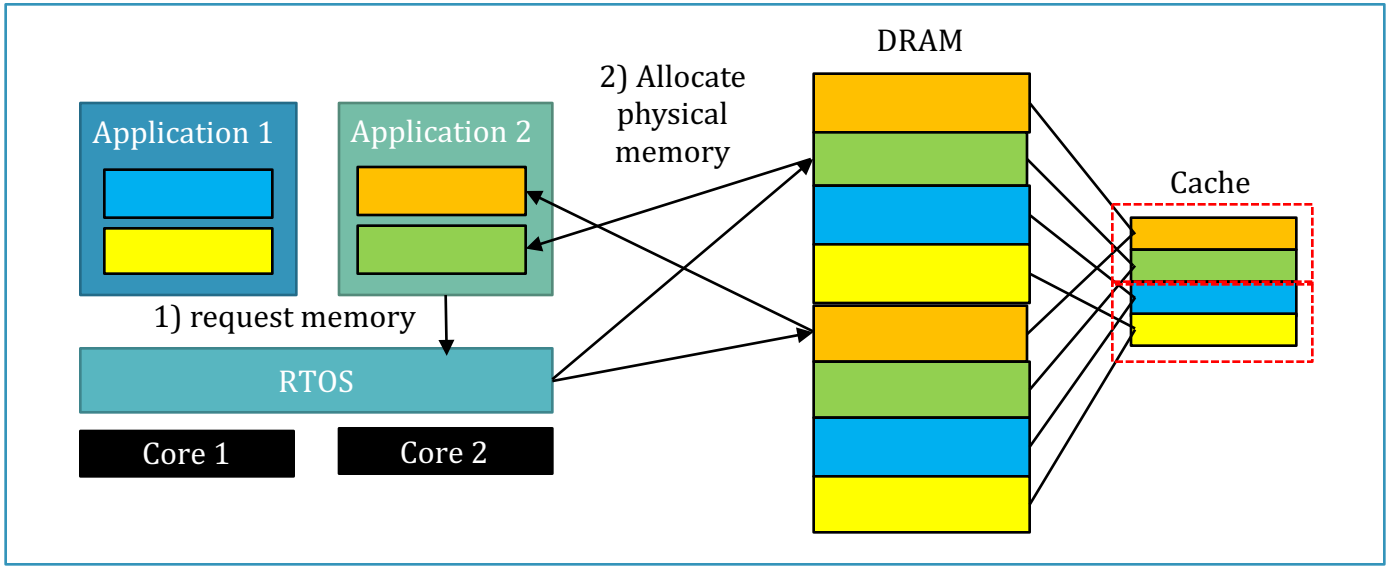
* H. Yun, R. Mancuso, Z.-P. Wu, and R. Pellizzoni. PALLOC: Dram bank-aware memory allocator for performance isolation on multicore platforms. In RTAS, 2014.

Resource partitioning mechanisms

Set-based cache partitioning [1/2]

- **Set-based partitioning via page coloring***
  - Allocation of physical memory considering the cache set location
  - $number\ of\ colors = \frac{cache\ size}{page\ size * cache\ assiciativity}$



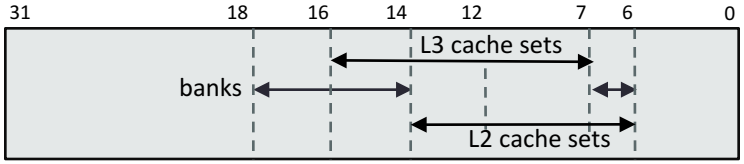


* R. Mancuso, R. Dudko, E. Betti, M. Cesati, M. Caccamo, and R. Pellizzoni. Real-time cache management framework for multi-core architectures. In RTAS, 2013.
* M. Chisholm, B. C. Ward, N. Kim, and J. H. Anderson. Cache sharing and isolation tradeoffs in multicore mixed-criticality systems. In RTSS, 2015.

Resource partitioning mechanisms

Set-based cache partitioning [2/2]

- **Implementations**
  - Manipulates virtual to physical address mapping – allocate disjoint cache sets to each core
    - Among address bits [15:7], cache set index, exploits [15:12] bits, which intersects with the physical page number in P4080
- **L2 co-partitioning & Restrictions of set-based partitioning**
  - Co-partition L2 cache
    - L3 cache set is determined by [15:12] and L2 cache set by [13:6]
    - Using [13:12] bits has a side effect of co-partitioning L2 cache
  - Only the [15:14] bits are allowed for L3 cache set partitioning
    - The number of cache partitions is limited to 4
    - If we adopt for 8 cores, some cache sets inevitably shared by 2 cores

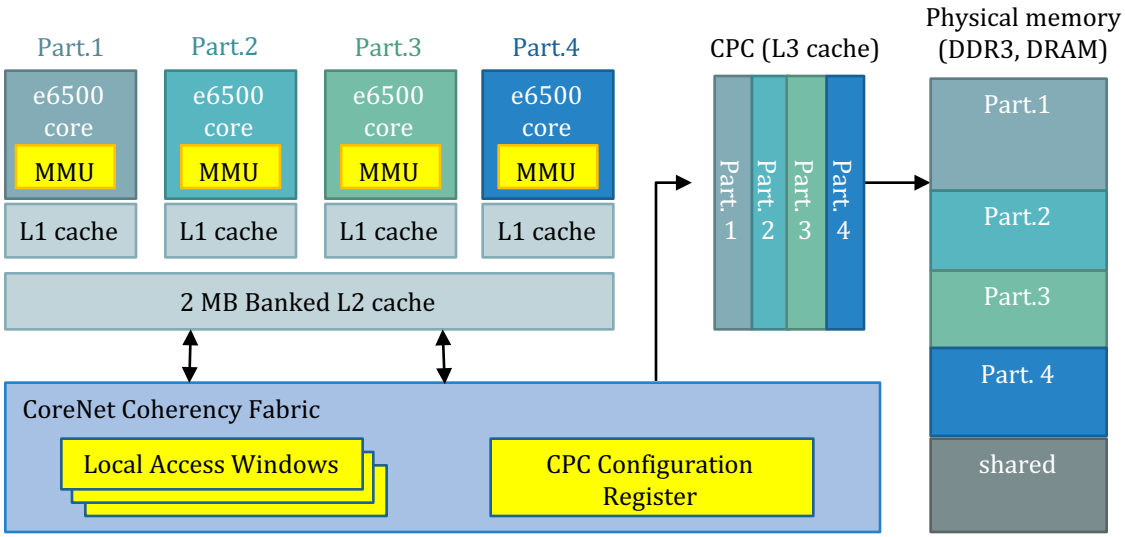


[P4080 memory address mapping]

Resource partitioning mechanisms

Way-based cache partitioning [1/2]

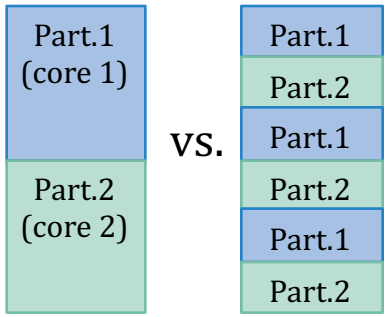
- **Way-based partitioning with Hardware-level support**
  - Configure main memory with multiple distinct partitions
    - For each partition, register the (memory range, target, and partition ID) in the LAW (Local Access Window) register
  - Partition the L3 cache and allocate disjoint cache ways to each core
    - Configure the L3 cache(CPC) related registers – transactions from the specified partition can allocate the blocks in the designated cache ways
      - E.g., transactions from the 'partition 1' allocate blocks in the 'way 0, 1, 2, 3'



Resource partitioning mechanisms

Way-based cache partitioning [2/2]

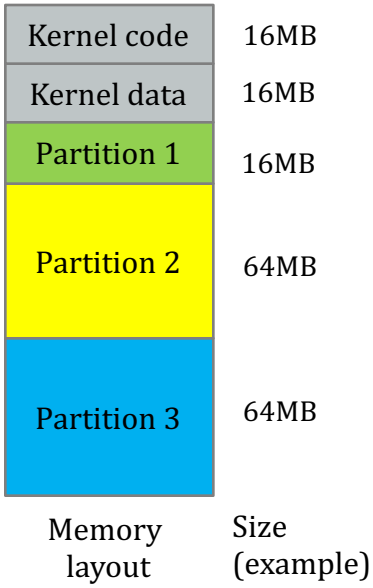
- **Relaxed restrictions on the number of cache partitions**
  - With set-based cache partitioning, number of cache partitions is restricted up to four
  - P4080 supports cache partitioning with per-way granularity, with each way providing 32KB
    - L3 cache is 32-way and can be partitioned to 32 parts
- **Limitations of way-based cache partitioning**
  - Way-based cache partitioning cannot be used with set-based cache or memory bank partitioning
  - Conflicting requirement of memory allocation
    - Sequential vs. interleaving
    - May be relevant to all other PowerPC chip models
  - Cache way locking allow integration
    - Most ARM processors supports cache way locking
    - PowerPC e500mc processor supports cache locking in a block granularity



Implementation issues

From the perspective of an RTOS [1/4]

- **Challenges – paging**
  - Page coloring assumes that OS manages memory with fixed-sized pages (normally, 4KB)
  - Qplus-AIR deliberately avoid paging due to the timing predictability is worsened when a TLB miss occurs within a paging scheme
- **Memory management of Qplus-AIR**
  - Managed with variable sized pages rather than fixed 4KB pages
    - Kernel data/code, partition regions
    - Manages each region as one large page
      - 1 TLB entry for each region
    - OS locks the entry in the TLB
      - Force all the mapping data to stay in the TLB
    - Size of memory for each application is configured by developers
  - MMU is used to prevent cross-application memory accesses

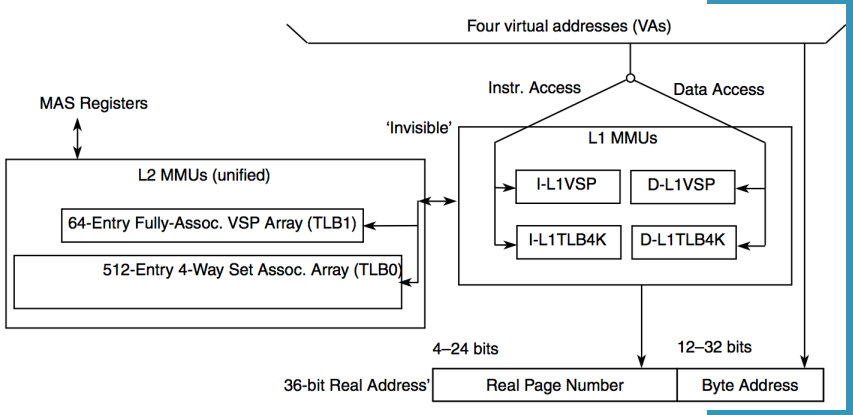


Implementation issues

From the perspective of an RTOS [3/4]

- **Memory management in P4080**
  - Two levels of MMU
    - Hardware-managed L1 MMU
    - Software-managed L2 MMU
  - Each MMU consists of
    - TLB for variable-sized pages (VSP), 11 different page sizes (4KB~4GB)
    - TLB for 4KB fixed-sized pages (FSP)
  - TLB locking for variable-sized pages

[ref.] PowerPC e500mc core reference manual

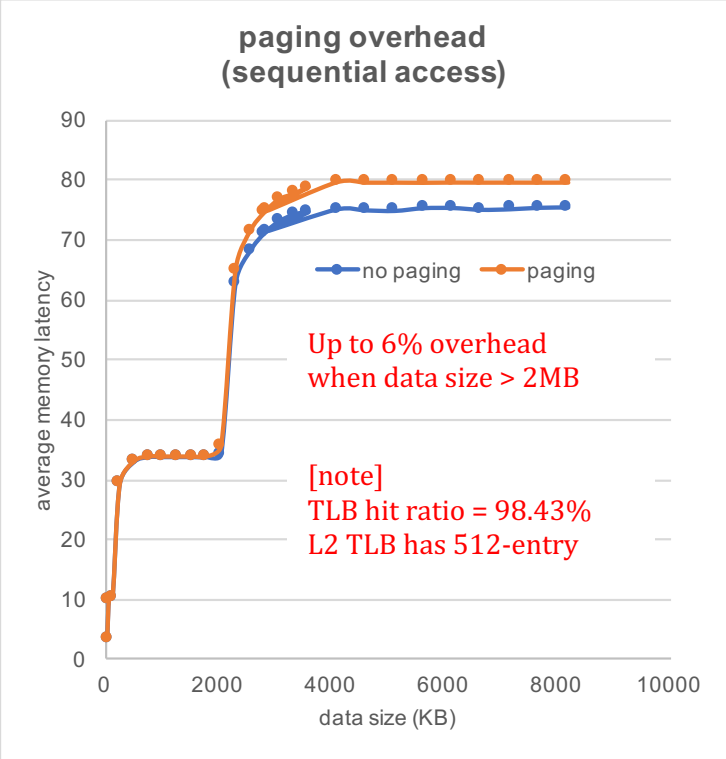


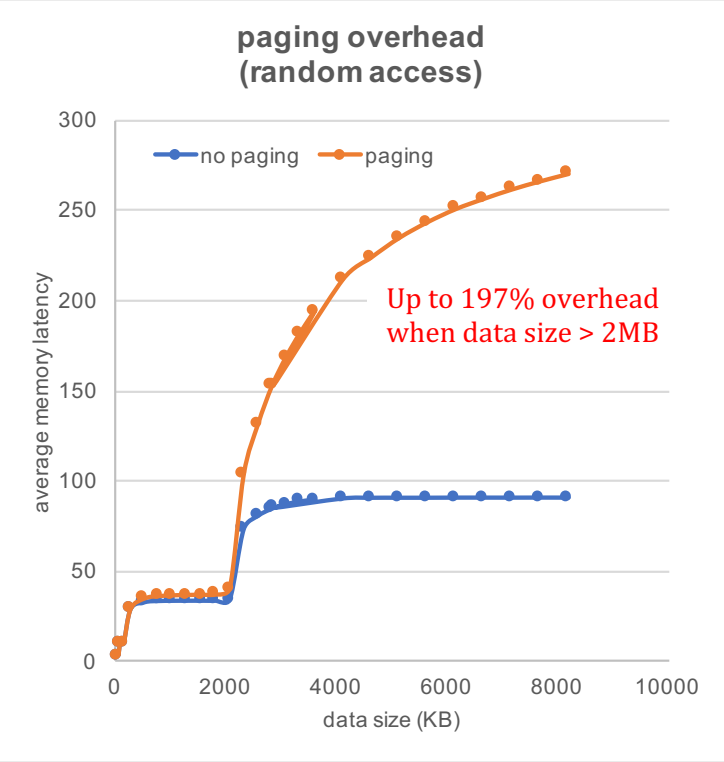
- **Modify memory management of Qplus-AIR**
  - To support page coloring, which is used to implement memory bank partitioning and set-based cache partitioning
  - Manage application's memory regions with 4KB granularity
    - Management of kernel regions was unchanged – bind performance predictability of kernel execution

Implementation issues

From the perspective of an RTOS [3/4]

- **Overhead of paging**
  - 'Latency' benchmark with changing data size and access pattern
    - Sequential access and random access of linked list
  - Measure the average memory access latency

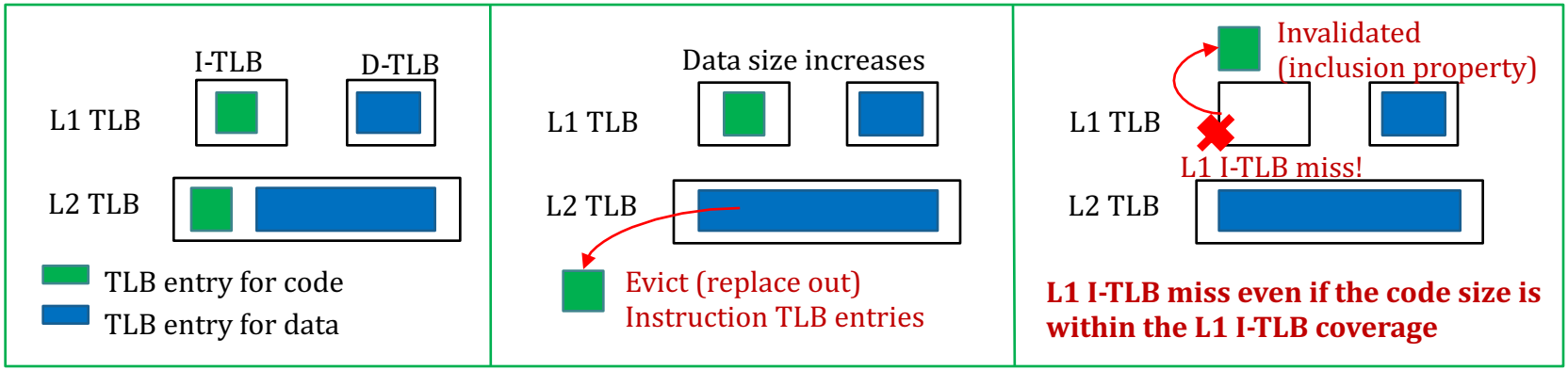




Implementation issues

From the perspective of an RTOS [4/4]

- **Analysis of overhead**
  - Degradation is due to the MMU architecture of e500mc core
    - L1 instruction and data TLBs and L2 unified TLB
    - L1 MMU is controlled as an inclusive cache of L2 MMU
    - In PowerPC e6500 core, L1 and L2 MMU is not inclusive



- **Requirements for the predictable paging**
  - Some studies focused on predictable paging*
  - COTS hardware provides means for implementing predictable paging – software-managed TLB or TLB locking

* D. Hardy and I. Puaut. Predictable code and data paging for real time systems. In ECRTS, 2008.

* T. Ishikawa, T. Kato, S. Honda, and H. Takada. Investigation and improvement on the impact of tlb misses in real-time systems. In OSPERT, 2013.

Resource partitioning mechanisms

Integration of partitioning schemes

- **Four techniques with paging**
  - Memory bus partitioning (RP-BUS), memory bank partitioning (RP-BANK), set-based cache partitioning (RP-$SET), and way-based cache partitioning (RP-$WAY)
- **Integration of memory bus, memory bank, and set-based and way-based cache partitioning mechanisms**
  - Note that way-based cache partitioning cannot be integrated with memory bank partitioning or set-based cache partitioning
- **Possible integration options**
  - Integration option #1 : RP_BUS, RP_BANK, and RP_$SET
    - Restrictions on the number of available cache partitions
  - Integration option #2: RP_BUS and RP_$WAY
    - Contentions on memory bank is unavoidable

Evaluations [1/5]

- **Evaluation setup**
  - Hardware platform
    - P4080 with activate 4 or 8 of total 8 cores
  - Software platform
    - Qplus-AIR
  - Synthetic benchmark
    - Latency : traverse a linked list to perform a read/write operation on each node, memory request is made one at a time
    - Bandwidth : access memory in sequence with no data dependency between conservative accesses – CPU generate multiple memory requests in parallel, maximizing memory level parallelism(MLP) available in the memory system
  - Metric
    - Average memory access latency (ns) – time to read/write one block (64B)
    - Normalize average latency to the best-case without resource contention

Evaluations [2/5]

- **Evaluation setup**
  - Two benchmark mixes
    - 4-core MIX
      - Cause contention on all the memory resources to evaluate each partitioning mechanism and integrated one
    - 8-core MIX
      - to show the limitation of set-based cache partitioning

Core 1	Core 2, 3	Core 4
Latency (512KB)	Bandwidth (4MB)	Bandwidth (32MB)

Core 1, 2	Core 3, 4, 5, 6	Core 7, 8
Latency (512KB)	Bandwidth (4MB)	Bandwidth (32MB)

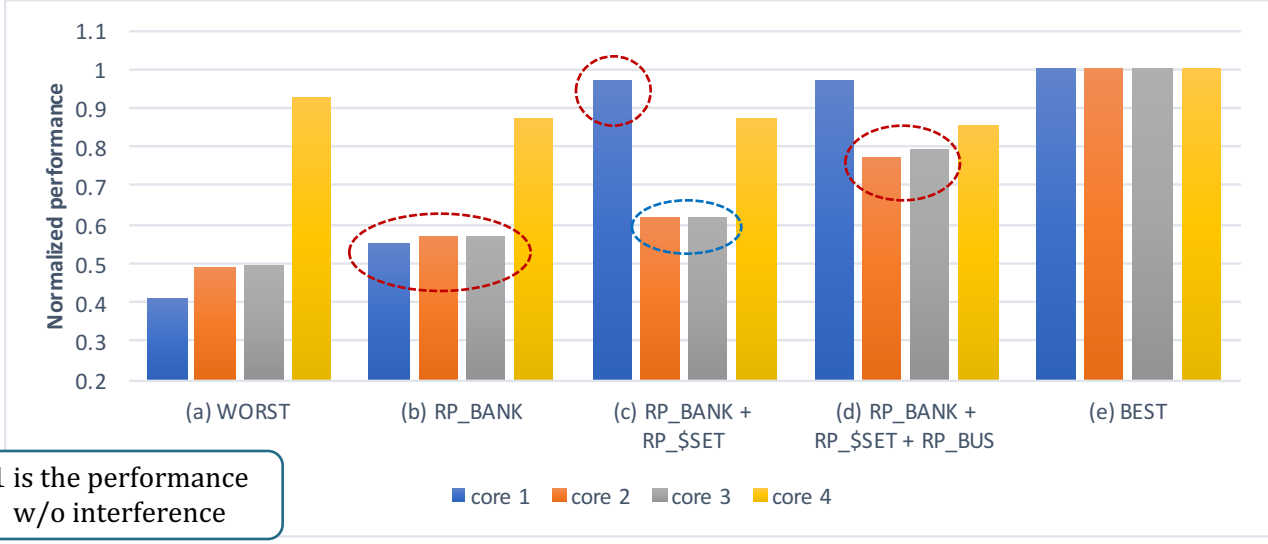
  - Data size configuration

	Data size	Examples Platform: 2MB LLC on 4-core CPU	Cache(LLC) hit rate
LLC	Size of LLC divided by number of cores	2MB / 4 cores = 512KB	100 %
DRAM/small	Twice the size of LLC	2MB ·2 = 4MB	0 %
DRAM/large	Significantly larger than LLC	Much larger than 2MB (32MB in our experimental setup)	0 %

Evaluations [3/5]

	(a)	(b)	(c)	(d)	(e)
core 1	0.41	0.55	0.97	0.97	1.00
core 2	0.49	0.57	0.62	0.78	1.00
core 3	0.50	0.57	0.62	0.79	1.00
core 4	0.93	0.87	0.87	0.85	1.00

- **4-core MIX, Integration Option #1**
  - RP_BANK, RP_$SET, and RP_BUS
  - (b) RP_BANK: all the cores are enabled to access banks in parallel
  - (c) Adding RP_$SET ensures 512KB L3 cache for Latency(LLC) app running on core1 (56% improvement compared to the worst-case)
    - Moreover, fewer accesses to main memory were requested by core1 helps performance on other cores
  - (d) Add RP_BUS: Performance when all techniques are put together

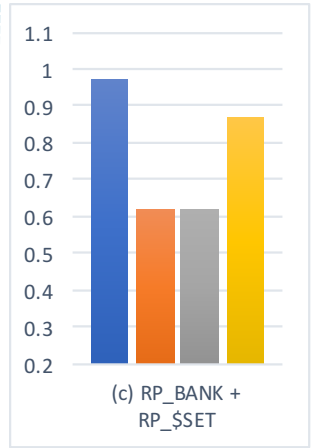


1 is the performance w/o interference

Evaluations [4/5]

	(a)	(b)	(c)	(d)
core 1	0.41	1.00	1.00	1.00
core 2	0.49	0.78	0.91	1.00
core 3	0.50	0.79	0.91	1.00
core 4	0.93	1.01	0.89	1.00

- **4-core MIX, Integration option #2**
  - RP_$WAY and RP_BUS
  - RP_BANK is inapplicable
    - In this benchmark, memory access is not concentrated to a bank since RP_$WAY allocates memory to each core sequentially
    - However, worst case could arise depending on an task behavior
  - RP_$WAY vs. RP_SET
    - Paging overhead on RTOS degrades performance
      - 3%, 16%, 17%, and 13% for each application on core 1, 2, 3, and 4

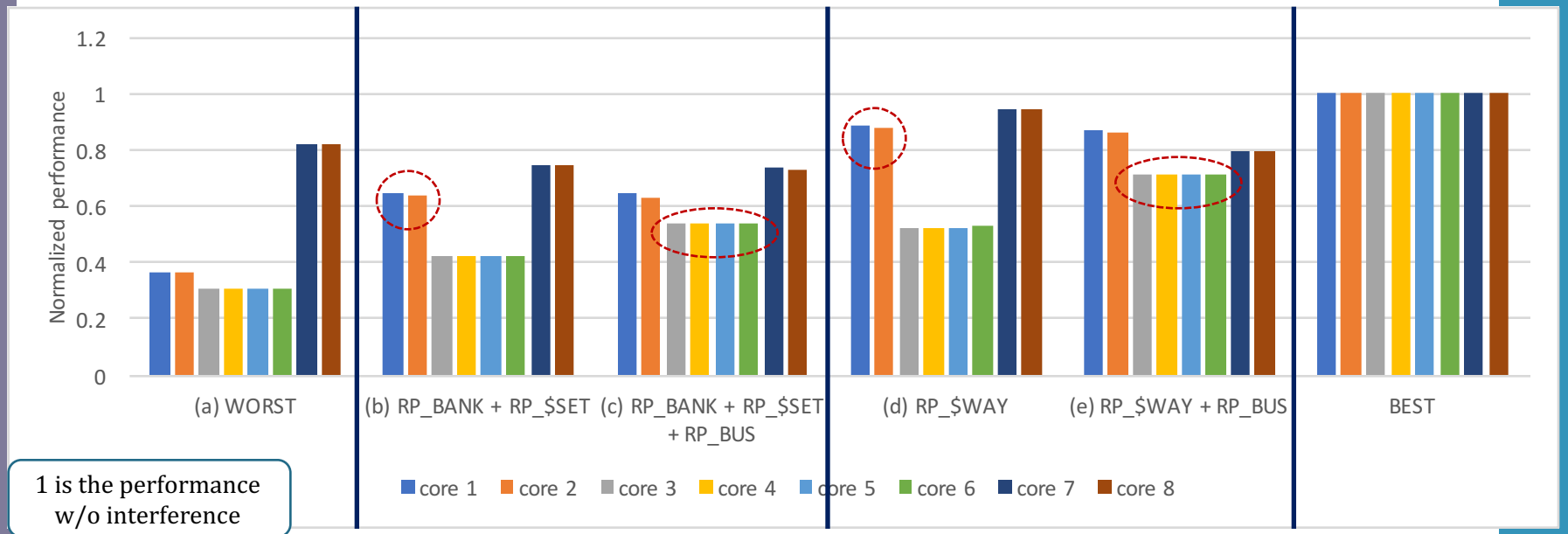


1 is the performance w/o interference

Evaluations [5/5]

	(a)	(b)	(c)	(d)	(e)	(f)
core 1	0.37	0.64	0.64	0.88	0.87	1.00
core 2	0.37	0.64	0.63	0.88	0.86	1.00
core 3	0.30	0.42	0.54	0.52	0.71	1.00
core 4	0.30	0.42	0.54	0.52	0.71	1.00
core 5	0.30	0.42	0.54	0.53	0.71	1.00
core 6	0.30	0.42	0.54	0.53	0.71	1.00
core 7	0.82	0.75	0.74	0.94	0.79	1.00
core 8	0.82	0.74	0.73	0.94	0.79	1.00

- **8-core MIX, Integration #1 & #2**
  - Restrictions on number of possible cache partitions
    - RP_$SET – 4 partitions, RP_$WAY – 32 partitions in P4080 platform
    - Performance of Latency(LLC) is about 64% and 88% with RP_$SET and RP_$WAY, respectively
  - Overhead of paging
    - Compare the performance in (b) and (d), or (c) and (e)



1 is the performance w/o interference

Conclusions & Future Work

- **Conclusions**
  - Qplus-AIR, an ARINC653 compliant RTOS
  - Comprehensive shared resource partitioning implementation on an ARINC653 compliant RTOS, Qplus-AIR
    - Implementation issues of implementing and combining multiple resource partitioning mechanisms
    - The unique challenges we encountered due to the fact that the RTOS did not support Linux-like dynamic paging
- **Future Work**
  - Predictable paging
  - Evaluation with real-world applications

Thank You
for the attention

pakeunji@etri.re.kr

References [1/2]

[1] Airlines Electronic Engineering Committee, Avionics Application Software Standard Interface ARINC Specification 653 Part 1, 2006.

[2] BIOS and kernel developer's guild for AMD family 15h processors, March 2012.

[3] ARM Cortex53 Technical Reference Manual, 2014.

[4] P4080 QorIQ Integrated Processor Hardware Specifications, Feb 2014.

[5] Certification Authorities Software Team, Position Paper CAST-32A : Multi-core Processors, 2016.

[6] QorIQ T2080 Reference Manual, 2016.

[7] M. Chisholm, B. C. Ward, N. Kim, and J. H. Anderson. Cache sharing and isolation tradeoffs in multicore mixed-criticality systems. In RTSS, 2015.

[8] J. Flodin, K. Lampka, and W. Yi. Dynamic budgeting for settling dram contention of co-running hard and soft real-time tasks. In SIES, 2014.

[9] D. Hardy and I. Puaut. Predictable code and data paging for real time systems. In ECRTS, 2008.

[10] T. Ishikawa, T. Kato, S. Honda, and H. Takada. Investigation and improvement on the impact of tlb misses in real-time systems. In OSPERT, 2013.

[11] H. Kim, A. Kandhalu, and R. Rajkumar. A coordinated approach for practical os-level cache management in multi-core real-time systems. In ECRTS, 2013.

[12] T. Kim, D. Son, C. Shin, S. Park, D. Lim, H. Lee, B. Kim, and C. Lim. Qplus-air: A do-178b certifiable arinc 653 rtos. In The 8th ISET, 2013.

References [2/2]

[13] R. Mancuso, R. Dudko, E. Betti, M. Cesati, M. Caccamo, and R. Pellizzoni. Real-time cache management framework for multi-core architectures. In RTAS, 2013.

[14] M.D.Bennett and N.C.Audsley. Predictable and efficient virtual addressing for safety-critical real-time systems. In ECRTS, 2001.

[15] J. Nowotsch and M. Paulitsch. Leveraging multi-core computing architectures in avionics. In EDCC, 2012.

[16] J. Nowotsch, M. Paulitsch, D. Bu¨hler, H. Theiling, S. Wegener, and M. Schmidt. Multi-core interference-sensitive wcet analysis leveraging runtime resource capacity enforcement. In ECRTS, 2014.

[17] S. A. Panchamukhi and F. Mueller. Providing task isolation via tlb coloring. In RTAS, 2015.

[18] M. K. Qureshi and Y. N. Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In MICRO, 2006.

[19] R.E.Kessler and M. D.Hill. Page replacement algorithms for large real-indexed caches. In ACM Trans. on Comp. Sys., 1992.

[20] L. Sha, M. Caccamo, R. Mancuso, J.-E. Kim, and M.-K. Yoon. Single core equivalent virtual machines for hard real-time computing on multicore processors, white paper. 2014.

[21] N. Suzuki, H. Kim, D. de Niz, B. Anderson, L. Wrage, M. Klein, and R. Rajkumar. Coordinated bank and cache coloring for temporal protection of memory accesses. In ICCSE, 2013.

[22] H. Yun, R. Mancuso, Z.-P. Wu, and R. Pellizzoni. Palloc: Dram bank-aware memory allocator for performance isolation on multicore platforms. In RTAS, 2014.

[23] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. Memory bandwidth management for efficient performance isolation in multi-core platforms. IEEE Transactions on Computers, 65:562–576, 2015.