STUDYING DIFFERENT TECHNIQUES FOR REDUCING INTERFERENCE IN MIXED-CRITICALITY SYSTEMS FOR MULTICORE PLATFORMS

JAVIER FERNANDEZ, TAMARA LUGO, JESUS CARRETERO

INDEX

CONTEXT

- Real-time applications have a huge impact in several fields.
  - i.e., Multimedia streaming applications, embedded applications for monitoring and controlling, etc.
- Real-Time requirements can be hard (no deadline missed) or soft (some deadline can be missed).
  - Different levels of criticality for different applications.
- Performance in Real-Time applications is secondary to fulfill the Real-Time requirements.
- Some environments requires the execution of mixed-criticality applications.
  - Including hard Real-Time applications, soft Real-Time applications and normal (best-effort) applications.
- Nowadays the execution of mixed-criticality workloads on multicore systems present some problems.
  - The amount of shared hardware between cores / applications creates interferences.
  - These interferences makes very difficult to ensure the fulfillment of the deadlines.

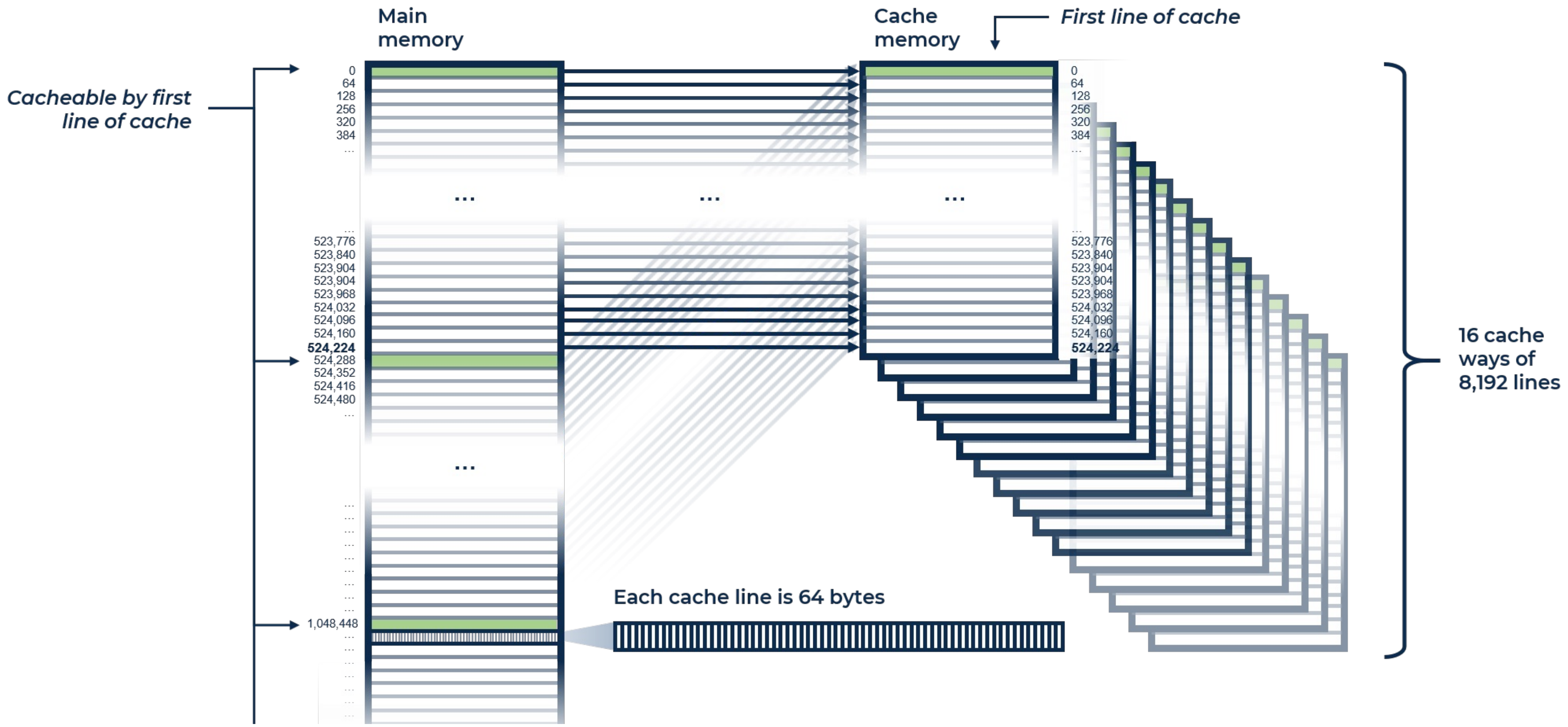
MOTIVATION

- The execution of mixed-criticality workloads on multicore nodes requires to reduce these interferences.
  - This is needed to ensure the deadlines while losing the least amount of performance possible.
- Real-Time applications are very dependent on the Worst-Case Execution Time (WCET).
  - The time that takes to execute when every resource behaves the worst (that includes interferences).
  - Real-Time applications can not be expected to run faster than this even though they could.
- Reducing interferences allows to obtain a better WCET.
  - However, it can reduce the performance of other applications and the overall systems.
- The goal is to select the correct techniques to reduce the interferences when executing a mixed-criticality workload in order to enhance the WCET the most without reducing the overall performance.

SOURCES OF INTERFERENCES IN MULTICORE PLATFORMS

- Memory interferences.
  - Exclusive access to each Memory Bank
    - Each Memory Bank is accessed through a row buffer.
  - DRAM Access controller schedules the access to each bank.
- Memory bus interferences.
  - A scheduling mechanism is needed to share it.
- Cache interferences.
  - Normally implemented as set associative caches with several ways.
  - Several levels of cache.
    - Core-exclusive caches vs. shared caches.

EXAMPLE: SET ASSOCIATIVE CACHE WITH 16 WAYS



TECHNIQUES TO REDUCE CACHE INTERFERENCES

- Cache-locking.
  - Prevent specific cache lines from being removed.
  - Allow a better computation of the WCET.
  - Only a few CPUS implement it (i.e., LEON4).
- Cache partitioning.
  - Split the cache for exclusive use of each core/process.
  - Two main techniques:
    - Hardware cache partitioning.
    - Cache coloring (software technique).

CACHE-LOCKING

- Cache-locking is a hardware feature in some CPU architectures (i.e., LEON4).
  - Allows to tag certain cache lines as non-removable.
  - This tags can be modified during the program execution.
- Cache locking is mostly used to improve the WCET for Hard Real-Time applications.
  - Content locked in cache can be accessed fast with total certainty.
  - However locked content reduce the cache effectiveness.
    - A balance of locked/free content is needed per application.

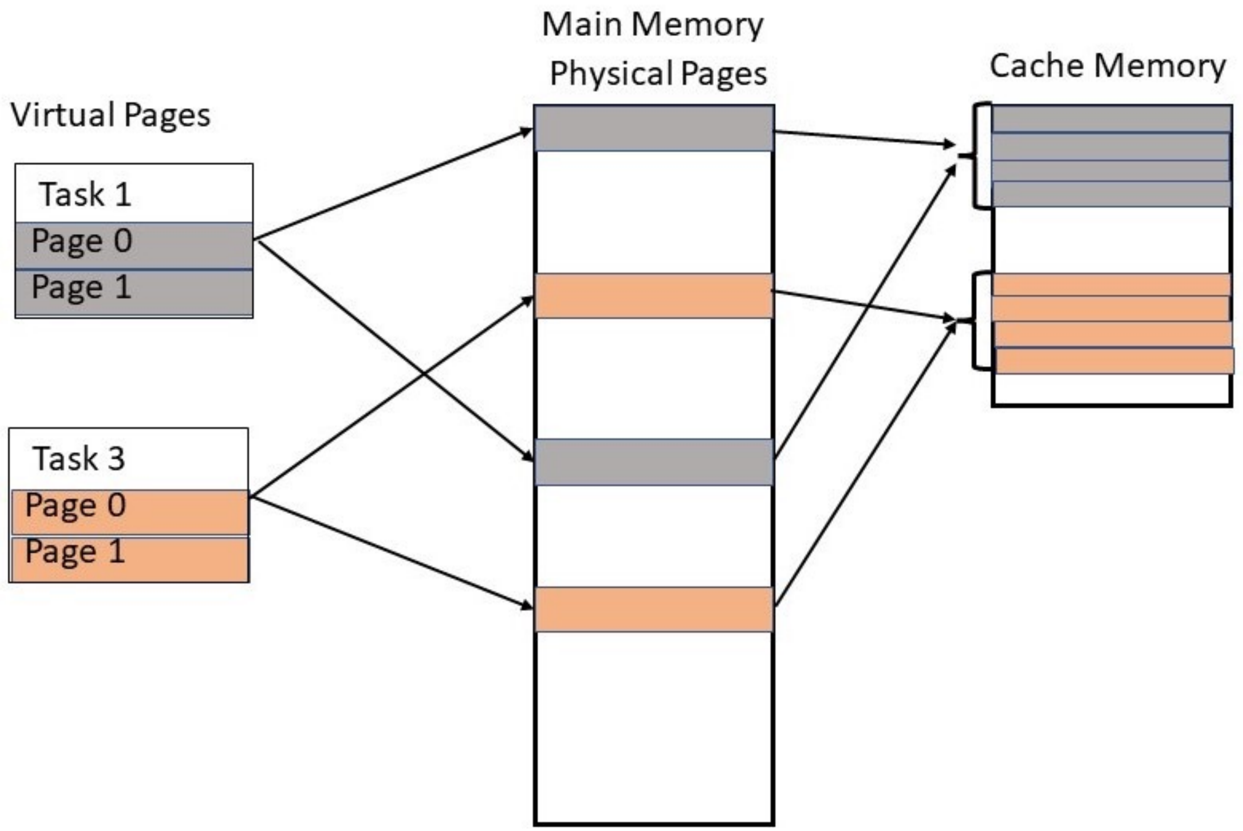
HARDWARE CACHE PARTITIONING

- Cache partitioning is the segmentation of the cache space.
  - Each partition is allocated for a certain task/core.
  - Partitions can be statically settled at the beginning or dynamically modified.
- Hardware cache partitioning is a feature on many recent CPUS (last 5 years).
  - Allows to partition shared caches, normally the last level cache (LLC).
    - Cache ways can be split in several groups each one with an ID.
  - Cache partitions can be allocated per core.
    - Each core can be assigned with one or several ID groups.
- Examples of Hardware cache partitioning architectures.
  - Intel: Cache Allocation Technology (CAT). Present in modern Xeon and Atom processors.
  - Arm: Memory System Resource Partitioning and Monitoring (MPAM). Present since Armv8.4-A architecture.

CACHE COLORING

- Cache-coloring is a software-only approach to cache partitioning.
- Based on a side-effect of set associative caches
  - Only a small number of cache lines can co-exist in the cache.
  - This co-exit lines always reside a multiple of the set size apart.
  - The maximum number of co-existing lines is equal to the number of ways in the cache.
- The technique divides memory into sections that can co-exist in cache.
  - Each memory section is given a color.
  - Each color is given to a unique task.
- The color assignation is done using virtual memory.
  - Cache lines on the same page always have the same color.
  - Each task is only assigned pages of the same set of colors.

CACHE COLORING



FRAMEWORK FOR REDUCING INTERFERENCES IN MIXED-CRITICALITY WORKLOAD

- Integrated Management for different processes categories.
  - Different execution schemas for each category.
  - Compute the expected performance for the size of each cache partition.
    - Calculate the WCTE for Hard RT applications.
    - Calculate a statistical performance for soft RT and best-effort applications.
- Propose a cache partition schema thar covers all levels of cache.
  - Each level can use different cache partition techniques.
- Improve the WCET by blocking cache data.
  - Using hardware cache-locking.
  - Propose a technique to block data on cache using cache coloring.

INTEGRATED MANAGEMENT FOR MIXED-CRITICALITY WORKLOADS

- The goal is to reduce interferences among applications of different criticality.
- Basic kinds of criticality.
  - Hard Real-Time applications: Missing one deadline is total failure.
  - Soft Real-time applications: Missing less deadlines than a certain value is acceptable.
  - Best-effort applications: No deadlines or real-time requirements needed.
- Applications can be executed on the cores as following:
  - One application execute exclusively on a single core.
    - Best option for Hard RT apps and best-effort applications that need performance.
  - Several applications can execute on a single core.
    - Using Real-Time scheduling algorithms (Hard and soft RT apps.).
    - Using normal scheduling algorithms (best effort apps.)

INTEGRATED CACHE PARTITIONING

- Cache partitioning is different depending on the level of cache.
- Shared cache: Two different options:
  - Hardware cache partitioning: Easy to implement, but only work on some (modern) CPUs.
  - Cache coloring: Works on any set associative cache but reduces the hit ratio from plain partitioning.
- Exclusive cache: No hardware support..
  - Cache coloring Is the last option. Only needed if:
    - More than one application per core.
    - Using cache coloring for other purposes (blocking cache data).

IMPROVE THE WCET BY BLOCKING CACHE DATA

- Hard Real-Time applications are limited by the Worst-Case Execution Time (WCET).
  - No matter if the data is in cache If there is a chance it could have been only in memory.
  - There are some techniques to statically analyze the code and ensure sometimes that certain data is in cache.
- One way to improve WCTE is to block in cache data that is more frequently used.
  - However, it reduces the probability of the rest of data to be on cache.
- Data can by block in cache using hardware features on certain CPUs.
- Alternative proposed: Using cache coloring:
  - Using more than one color per application and use one color only for blocking data on cache.

EVALUATION INFRASTRUCTURE

- Design of a simulation framework.
  - Simulation of a multilevel set associative cache.
- Design a mixed-criticality workload.
  - Schema to obtain execution logs with memory operation and addresses.
- Compare several management techniques to avoid interferences.
  - Implement the integrated management for mixed-criticality workloads.
  - Compare different implementations using different techniques (cache-coloring, hardware partitioning, cache-locking).

DESIGN OF A SIMULATION FRAMEWORK

- A simulation framework is developed for:
  - Simulate execution of computing traces generated from real executions.
  - Simulate the virtual memory mapping for implementing cache coloring.
  - Simulating several levels of cache (set associative caches with several ways) that featured.
    - Hardware cache partitioning.
    - Hardware cache-locking.
- The simulation Framework is implemented using MATLB/SIMULINK.
  - Traces are obtained from real execution onto ARM CPUs.

DESIGN A MIXED-CRITICALLITY WORKLOAD

- Designing a number of workloads including several application traces with different criticality.
  - Changing criticality levels, deadlines margin, computation/memory intensity, etc.
- Traces are generated from real execution logs.
  - Using GDB to obtain an execution log that includes:
    - Memory address of the instruction.
    - Assemble code from the instruction (including operands).
  - GDB also allows to obtain memory addresses for data operands.

COMPARE SEVERAL MANAGEMENT TECHNIQUES TO AVOID INTERFERENCES

- Different experiments are designed to test different configurations:
  - Workloads including different applications with different levels of criticality.
  - Different cores assignation to single/multiple applications.
  - Different cache partitioning techniques on each cache levels.
  - Using cache-locking techniques for improving WCET.
- The goal is to obtain a general strategy that:
  - Improve general performance.
  - Improve WCET for Hard Real-Time applications.

STATE OF THE WORK

- Complete / almost complete.
  - The simulation framework presented is in the final states of completion.
  - A tool to automatically generated execution traces is already developed.
  - Several workloads of application traces are already generated.
- To be done.
  - Implement and execute several mixed-criticality configurations using the finished simulation framework.
  - Compare the results using each combination of techniques for each workload.
  - Develop a general strategy to improve performance based on the obtained results.

CONCLUSIONS

- We have studied several techniques to reduce interferences on mixed-criticality workloads.
  - Mainly based on cache partitioning and cache-locking.
- An integrated approach is proposed to cover:
  - All the different levels of cache.
  - Workloads including applications with different levels of criticality.
- A simulation framework is developed to test all the proposed configurations.
  - Using traces generated from actual executions of the applications.
- Once the results are obtained a general strategy to improve performance will be derived from them.

PUBLICATIONS

- T. Lugo, S. Lozano, J. Fernández and J. Carretero.
  A Survey of Techniques for Reducing Interference in Real-Time Applications on Multicore Platform
  in *IEEE Access*, vol. 10, pp. 21853-21882, 2022, doi: 10.1109/ACCESS.2022.3151891.

STUDYING DIFFERENT TECHNIQUES FOR REDUCING INTERFERENCE IN MIXED-CRITICALITY SYSTEMS FOR MULTICORE PLATFORMS

JAVIER FERNANDEZ, TAMARA LUGO, JESUS CARRETERO