# Isolating Real-time from Processor-intensive Processes in Embedded Multi-core Systems

Piet Cordemans, Nico De Witte, Jens Vankeirsbilck, Willem Melis and Jeroen Boydens

*Abstract* – **Multi-core embedded systems allow to isolate critical processes on a CPU, while management is provided by a single operating system. This provides an opportunity to address real-time and security issues. We try to determine if processes can be effectively isolated under heavy processor load in a setup with the *Radxa*, a quad-core embedded system, and Linux *cgroups*. In our experiment we measure the variance of a time-critical signal while stressing the system. We explore different scenarios and determine the optimal configuration.**

*Keywords* – **Embedded operating systems, multi-core, soft-realtime**

## I. Introduction

Linux is in essence a time-sharing, multi-user operating system, which is focused on maximizing throughput. In order to prevent starvation due to prioritization, fairness is guaranteed in scheduling processes. Although a high-throughput of processes is a desirable property of an OS, Linux cannot preempt processes when executing system calls, or preempt kernel processes. Moreover the difference between minimal and maximal response time of a processes, this is known as jitter, cannot be limited for certain time-critical processes. In effect Linux sacrifices deterministic scheduling and cannot guarantee real-time behavior in favor of throughput and focusing on usability from a user perspective [1].

In contrast, a Real-Time Operating System (RTOS) favors deterministic scheduling and minimizes jitter for real-time critical processes, which are desirable properties when dealing with embedded hardware.

Nevertheless, embedded Linux is the favored OS for embedded systems despite the lack of real-time features.

### *A. Multi-core organization*

As multi-core embedded systems are adopted, the organization of CPU's offer opportunities to isolate time-critical from data-intensive processes. This is called the planar pattern, in which a control plane and data plane are defined [2]. By assigning the necessary hardware resources to the control plane, the time-critical processes are no longer interfered by the processes in the data plane, which utilize the other hardware resources. In order to optimize the utilization of hardware all resources which are not needed in the control plane are assigned to the data plane.

P. Cordemans, J. Vankeirsbilck and J. Boydens are with the Faculty of Engineering Technology, KU Leuven e-mail: {Piet.Cordemans, Jens.Vankeirsbilck, Jeroen.Boydens}@kuleuven.be

N. De Witte is with the Department of Electronics-ICT, Vives University college email: Nico.Dewitte@vives.be

W. Melis is a student of the Faculty of Engineering Technology, KULeuven.

The planar pattern can be applied in two general hardware configurations. On the one hand is the Symmetric Multi-Processing (SMP) organization, which involves a single OS to manage all available cores. In a SMP configuration all memory is shared which facilitates communication between processes. Furthermore it allows for an optimal use of resources and is flexible when more processes are added to the system.

On the other hand is Asymmetric Multi-Processing (AMP) organization, in which no longer a single OS manages all cores. Different configurations are possible to assign operating systems, real-time operating systems or no OS at all to a single CPU or a group of CPU's. In this setup isolating processes can be achieved by assigning these processes to separate operating systems.

In this paper we evaluate a technique called *cgroups* to isolate time-critical processes in a SMP organization. This should allow to combine the benefits of a SMP architecture with isolation of real-time critical processes.

Several approaches of adopting real-time behavior have been examined for embedded Linux. Papaux et al. [3] describe virtualization on an embedded system with KVM and measure performance degradation. However they do not consider jitter and variability in real-time critical processes. They rather focus on throughput of the virtualized system.

Varanasi and Heiser [4] developed a hypervisor which supports the hardware extensions of ARM. They conclude that virtualization extensions reduce hypervisor complexity and make it a viable approach with minimal instruction cycle overhead. They mention real-time support with a real-time clock which is configured as a pass-through device. However their implementation does not support a multi-core architecture.

Brandt et al. [5] describe a scheduler which supports a mix of best-effort, hard- and soft-real time processes. This scheduler is not compatible with the Linux operating system and does not support multi-core processing.

Considering reliability, two approaches have been examined. On the one hand Lee et al. [6] proposed a task remapping technique in order to increase reliability. It depends heavily on an extensive compile-time analysis.

On the other hand in [7] Huang et al. implemented a fault-tolerant task scheduling system for multiprocessor embedded systems. It considered both transient and permanent problems and optimally uses hardware and software redundancy.

Brandenburg et al. [8] presented a case study which evaluates different real-time scheduling algorithms.

This paper is organized as followed. In Section 2 the principles of *cgroups* are introduced. Next in Section 3 the setup and results of the experiments are given. In Section 4 the results are discussed. Finally future work is indicated.

## II. CGROUPS

*Cgroups* is a Linux kernel feature since version 2.6.24 and has had a major redesign since version 3.14. It allows to limit, prioritize and isolate CPU usage and other resources such as memory and I/O for a collection of processes.

### *A. CPUsets*

*CPUsets* is a part of *cgroups* which maps processes to specific banks in memory or to a logical CPU or set of logical CPU's. They extend process scheduling with a set of hooks in order to manage dynamic process scheduling on multi-core systems.

*CPUsets* communicate through the file system rather than using system calls. A virtual file system is mounted in which a hierarchical system of *CPUsets* is described. *CPUsets* are hierarchical in two ways. A child spawned of a process in a CPUset, automatically belongs to that CPUset. Moreover, a child of a CPUset is limited to the resources assigned to its parent. When a CPUset consists only of resources which are exclusively assigned to it and its children, this CPUset is called an exclusive CPUset.

### *B. Scheduling*

*CPUsets* extend the behavior of the scheduling system by a set of hooks, which manipulate process attributes such as affinity, real-time priority and the nice factor.

The load balancer is responsible for assigning processes to specific cores. Each process has an affinity attribute which binds the process to a specific queue of a core. Affinity is important to avoid trashing the cache by switching a process from one core to another all the time. When a specific process is isolated on a specific core, *CPUsets* manipulate all affinity attributes so that the load balancer assigns the specific process to the queue of the chosen core, while other processes are assigned to the queues of other cores.

The scheduler assigns time slices in a time window to the specific processes in its queue. It assigns a virtual runtime variable, which indicates how much of the processor time has been used by the process. As Linux employs a fair scheduler in the default scheduling policy, this means that the process with the lowest virtual runtime value has a higher priority. However there is the minimum granularity, which indicates the minimum time that is assigned to a process in the window regardless of its priority.

Setting the nice factor also changes the time which is assigned to a specific process. This attributes to a weight, which is relative to all the weights of all processes in the queue of the scheduler. Increasing the nice factor will result in a lower priority and vice versa.

Changing the real-time priority factor immediately interferes with scheduling the next process.

Finally, *CPUsets* do not guarantee exclusive access to the variables which manipulate priority of tasks. However if another system tries to manipulate priority while *CPUsets* are active, the system call is intercepted and an error is returned which indicates that *CPUsets* will maintain priority.

## III. EXPERIMENTS

Four experiments are conducted in different scenarios. One experiment without process isolation and three experiments with isolation of a time-critical process from CPU bound processes in different configurations. The general setup of the experiments is shown in Figure 1.

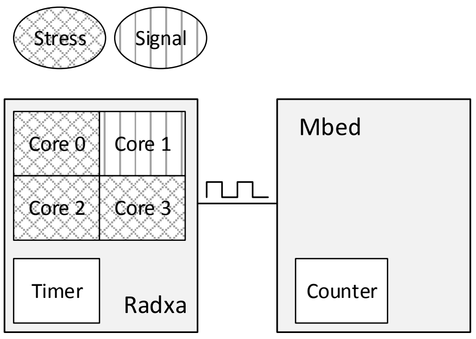

Fig. 1. Setup measurements of the time-critical signal

These have been conducted on the *Radxa Rock Pro* board which is built with the RK3188, a quad-core ARM Cortex-A9 processor [9]. This processor has a maximum clock frequency of 1.6GHz, all cores contain a L1 data and instruction cache of 32 KB and the processor has a shared 512 KB unified L2 cache. The board provides 2 GB of memory.

In our experiments we used a *Debian Wheezy* distribution based on the Linux 3.18 kernel. The *cgroup* package was obtained through the *aptitude* package manager.

The *Radxa* runs two programs. On the one hand, it generates a "time-critical" square wave signal with a period of 200 ms. This signal is generated through a real-time POSIX timer with a resolution of 1 ns. The timer runs asynchronously from the processor and notifies the timer trigger event to the handler process. On the other hand, the *Radxa* runs a CPU workload generator called *stress*, which spawns threads and waits for them to complete [10].

Each edge of the time-critical signal triggers the counter of an independent microcontroller, the *Mbed* LPC1768 [11]. The counter allows to measure the interval of the edges of the time-critical signal with resolution of 40 ns. This measurement is compared with the nominal value of 100 ms:

$$\Delta t_n = |t_n - t_{nominal}| \qquad (1)$$

As the time-critical signal represents a task which needs to adhere to real-time constraints, values $\Delta t_n$ larger than 0 represent the latency in scheduling that task. Variance in latency is expected as scheduling issues arise in the different scenarios.

In a real scenario real-time constraints are determined by the application, however in this case these are arbitrarily determined as follows. A context switch on this hardware is expected to take a couple of microseconds. When taking the extreme CPU-bound stress into account the range in which we want to observe variance is up to a factor 4 of a normal context switch. Therefore in this case when jitter exceeds 15µs the value is considered as an extreme value.

In each scenario 2048 batches of 50 edges are counted. All data and artefacts of these experiments are available at *github* [12].

### A. Baseline

In the baseline experiment no processes are isolated. In a first scenario no stress is applied. Figure 2 shows the discrete probability distribution function (pdf) of Δt.

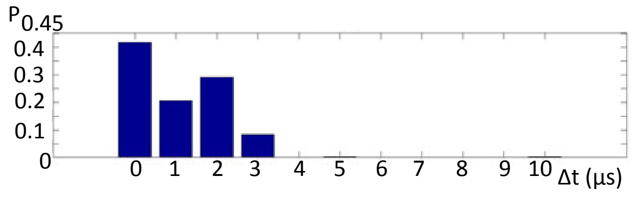

Fig. 2. No isolation, no stress: sample mean $\bar{y} = 0.0034\mu s$ and sample variance $\sigma_y^2 = 2.1461$

In a second scenario in the experiment without process isolation, CPU-bound stress is applied. I.e. 16 threads which executes an infinite loop calculating the square root of a single variable. The sample mean of Δt in this scenario is 327.5195 µs, a value larger than the period of the signal. This effect is due to extreme values in 26 of the 2048 samples, of which 14 are in the millisecond range. Figure 3 shows the discrete pdf of Δt smaller than 15 µs in this scenario.

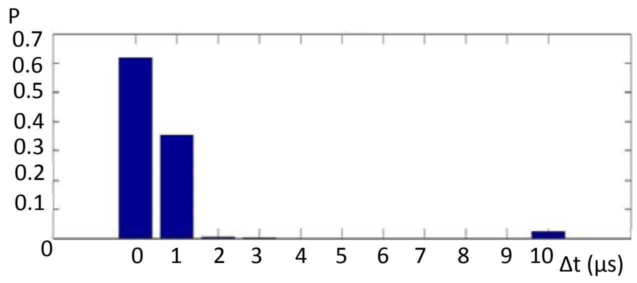

Fig. 3. No isolation, with CPU-bound stress, 16 threads, values of Δt < 15µs: $\bar{y} = 0.6048\mu s$ and $\sigma_y^2 = 0.2568$

### B. Isolating core 1

In this experiment, core 1 is isolated in an exclusive CPUset and only the time-critical process is assigned to it. The other cores handle the 16 threads of the CPU intensive process. Figure 4 shows the discrete pdf of Δt in this scenario. There were no extreme values.

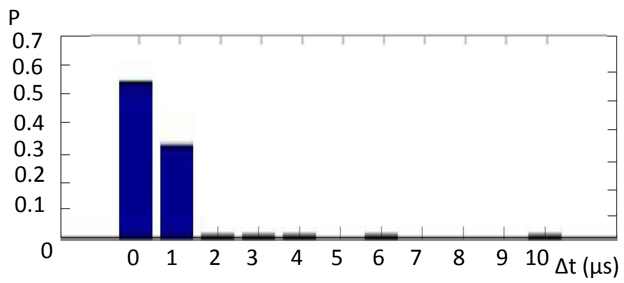

Fig. 4. Core 1 isolated, with CPU-bound stress, 16 threads, values of Δt: and $\sigma_y^2 = 0.2943$

### C. Isolating core 0 and 1

This experiment isolates both core 0 and core 1in an exclusive CPUset with only the time-critical process assigned to it. Core 2 and 3 execute 16 threads of the CPU intensive process. Figure 5 shows the discrete pdf of Δt. There was a single extreme value (30µs).

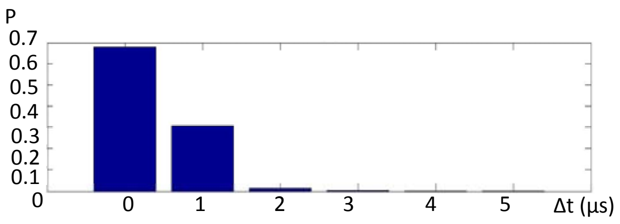

Fig. 5. Core 0 and 1 isolated, with CPU-bound stress, 16 threads, values of Δt < 15µs: and $\sigma_y^2 = 0.2523$

### D. Isolating half of core 1

In the fourth experiment a timeshare equal to half of all time available on core 1 is attributed to the time-critical process. The other cores and remaining slots on core 1 are executing the stress process. In this scenario there are 503 extreme values larger than 100 µs with maximum values in the range of 10 ms.

## IV. Discussion

When considering only time-critical processes which do not exceed a jitter of 15µs, the baseline experiment indicates a larger sample mean value and a smaller sample variance in the scenario with CPU-bound processes. In a standard configuration, process preemption allows scheduling in a deterministic fashion in almost 99% of the time. However depending on a situation where the time-critical task is interleaved and scheduled frequently, the expected jitter is unacceptable.

Isolating core 1 allows to fully utilize the other cores while isolating the time-critical process fully on a hardware level. Core 1 has been chosen as core 0 deals with OS related interrupts which might interfere with the time-critical process. However core 2 or 3 are equivalent in this respect. In this scenario, the sample mean is lower when compared to the baseline experiment with CPU-bound stress, while sample variance is higher. Nevertheless isolating core 1 prevents extreme jitter an indication that considering all sample values in this case accounts for the higher sample variance.

As core 0 shares memory caches with core 1, as well as deals with OS interrupts, the effect of isolating both cores considers a more prudent approach than isolating only a single core. Both sample mean and variance are lower, however a single extreme value has been measured. We conducted 6 additional experiments with 102 400 samples while isolating respectively core 1 and both core 1 and 0. These experiments indicated that extreme values occur in both scenarios once, never exceeding a maximum value of 30 µs.

The final experiment isolated half of the available time on core 1. This resulted in a considerable worse scenario than the baseline experiment with CPU-bound stress or the other isolation experiments. This is due to the implementation of attributing time shares of the core to a specific process. Whereas isolation of a core is achieved by manipulating the load balancer, assigning half of the time is the responsibility of the scheduler. However the scheduler can only perform a context switch when a tick occurs. As the ticks have a period of 10 ms, any real-time system requiring jitter lower than this value cannot rely upon this feature of *cgroups*. In essence the granularity of time slices is too large to reliably schedule the time-critical process every 100 μs.

## V. Future Work

In our experiments we applied a CPU-bound load in order to stress the OS and hardware system. However, CPU-bound congestion might not be the only cause of failing to meet real-time criteria. When the memory busses are saturated, the virtual memory system might fail to load the pages associated with the time-critical process, causing extra jitter. In a future experiment we will force a memory bus congestion and check the effect of process isolation on shared caches and main memory. Furthermore we can apply a combined set of processes, which induce CPU-bound as well as memory-bound stress.

Further, we might consider virtualization solutions for process isolation, like KVM and LXC. Virtualization is a technique popular in server environments, so the specific real-time constraints of an embedded system process need to be mapped onto these solutions. These techniques could be comparatively evaluated with *Cgroups*.

## VI. Conclusion

In this paper we evaluated the performance of *cgroups* on a multi-core embedded system. *Cgroups* are used to isolate critical real-time processes under CPU-bound stress in a SMP architecture. They allow to define *CPUsets* which manipulate the load balancer and schedulers. In a configuration which isolates the real-time process to an exclusive CPUset we conclude that process isolation is effective and considerably reduces jitter when compared with a configuration without *cgroups* enabled. When the scheduler is manipulated, for instance when trying to assign half of a window of a CPU to the critical task, jitter increases immediately up to the size of a time slice.

## References

[1] Love, Robert. Linux kernel development. Pearson Education, 2010.
[2] De Witte, Nico, Robbie Vincke, Sille Van Landschoot, Eric Steegmans, and Jeroen Boydens. "Evaluation of a dual-core SMP and AMP architecture based on an embedded case study." CW Reports, 2013.
[3] Papaux, Geoffrey, Daniel Gachet, and Wolfram Luithardt. "Processor virtualization on embedded linux systems." In Education and Research Conference (EDERC), 2014 6th European Embedded Design in, pp. 65-69. IEEE, 2014.
[4]Varanasi, Prashant, and Gernot Heiser. "Hardware-supported virtualization on ARM." In Proceedings of the Second Asia-Pacific Workshop on Systems, p. 11. ACM, 2011.
[5] Brandt, Scott, Scott Banachowski, Caixue Lin, and Timothy Bisson. "Dynamic integrated scheduling of hard real-time, soft real-time, and non-real-time processes." In Real-Time Systems Symposium, pp. 396-407. IEEE, 2003.
[6] Lee, Chanhee, Hokeun Kim, Hae-woo Park, Sungchan Kim, Hyunok Oh, and Soonhoi Ha. "A task remapping technique for reliable multi-core embedded systems." In Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, pp. 307-316. ACM, 2010.
[7] Huang, Jia, Jan Olaf Blech, Andreas Raabe, Christian Buckl, and Alois Knoll. "Analysis and optimization of fault-tolerant task scheduling on multiprocessor embedded systems." In Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, pp. 247-256. ACM, 2011.
[8] Brandenburg, Björn B., John M. Calandrino, and James H. Anderson. "On the scalability of real-time scheduling algorithms on multicore platforms: A case study." In Real-Time Systems Symposium, 2008, pp. 157-169. IEEE, 2008.
[9] RK3188 Technical Reference Manual. Rockchip, 2013.
[10] Stress, a deliberately simple workload generator. Available at: http://people.seas.harvard.edu/~apw/stress/
[11] Mbed NXP LPC1768 prototyping board.
[12] https://github.com/Zilleplus/thesis